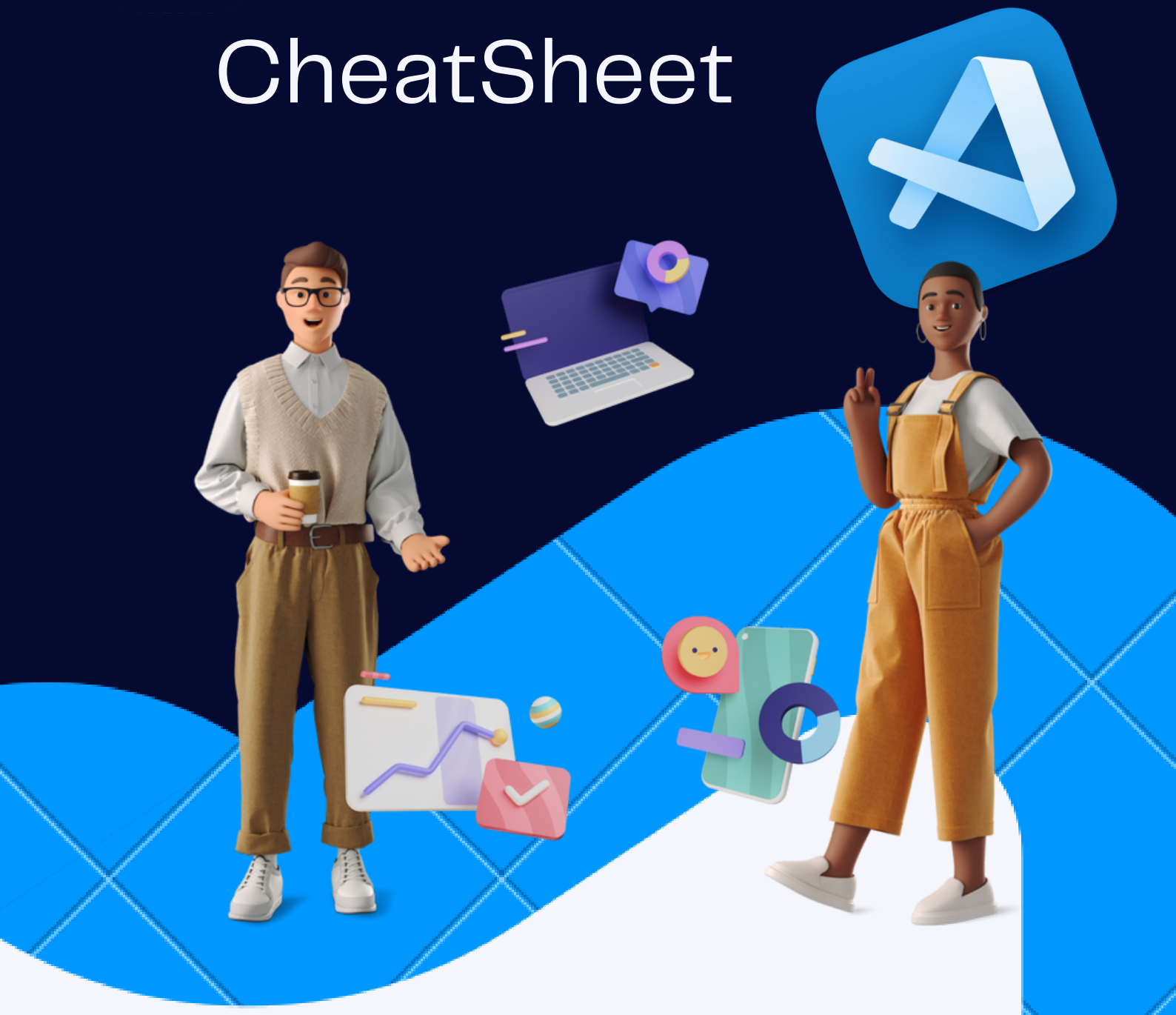
CODE HELP

VS Code

CheatSheet



CODE HELP

BASICS

Ctrl+Shift+N
New window/instance

Ctrl+Shift+W
Close window/instance

Ctrl+Shift+P or F1
Show Command Palette

Ctrl+P
Quick Open, Go to File

Ctrl+,
User Settings

Ctrl+K Ctrl+S
Keyboard Shortcuts

Ctrl+C
Copy line (empty selection)

Ctrl+X
Cut line (empty selection)

Alt+ ↑ / ↓
Move line up/down

Shift+Alt + ↓ / ↑
Copy line up/down

Ctrl+Shift+K
Delete line

Ctrl+Shift+Enter
Insert line above

Ctrl+Home
Go to beginning of file

Ctrl+End
Go to end of file

Home / End
Go to beginning/end of line

Ctrl+↑ / ↓
Scroll line up/down

Ctrl+Shift+
Jump to matching bracket

Ctrl+Enter
Insert line below Basic editing

Ctrl+] / [
Indent/outdent line

Ctrl+Shift+[
Fold (collapse) region

Ctrl+Shift+]
Unfold (uncollapse) region

Alt+PgUp / PgDn
Scroll page up/down Basic editing

Ctrl+/
Toggle line comment

Alt+Z
Toggle word wrap

Ctrl+K Ctrl+U
Remove line comment Navigation

Ctrl+T
Show all Symbols

Ctrl+P
Go to File...

Ctrl+K Ctrl+[
Fold (collapse) all subregions

Ctrl+K Ctrl+0
Fold (collapse) all regions

Ctrl+K Ctrl+J
Unfold (uncollapse) all regions

Ctrl+K Ctrl+]
Unfold (uncollapse) all subregions

Ctrl+K Ctrl+C
Add line comment

Ctrl+Shift+O
Go to Symbol

Ctrl+G
Go to Line Navigation

Ctrl+Shift+M
Show Problems panel

Shift+F8
Go to previous error or warning

Ctrl+Shift+Tab
Navigate editor group history

F8
Go to next error or warning

SEARCH AND REPLACE

Ctrl+F
Find

F3 / Shift+F3
Find next/previous

Alt+Enter
Select all occurences of Find match

Ctrl+H
Replace Search and replace

Ctrl+D
Add selection to next find match

Alt+C / R / W
Toggle case-sensitive / regex / whole word

Ctrl+K Ctrl+D
Move last selection to next find match

MULTI CURSOR SELECTION

Alt+Click
Insert cursor

Ctrl+U
Undo last cursor operation

Shift+Alt+I
Insert cursor at end of each line selected

Ctrl+Alt+ ↑ / ↓
Insert cursor above / below

Ctrl+L
Select current line

Ctrl+F2
Select all occurrences of current word

Shift+Alt+ →
Expand selection

Ctrl+Shift+L
Select all occurrences of current selection

Shift+Alt+ ←
Shrink selection

Ctrl+Shift+Alt + (arrow key)
Column (box) selection

Ctrl+Shift+Alt + PgUp/PgDn
Column (box) selection page up/down

Shift+Alt + (drag mouse)
Column (box) selection

LANGUAGE EDITING

F2
Rename

F12
Go to Definition

Ctrl+K F12
Open Definition to the side

Ctrl+.
Quick Fix

Alt+F12
Peek Definition

Ctrl+Space, Ctrl+I
Trigger suggestion

Shift+Alt+F
Format document

Ctrl+K Ctrl+F
Format selection

Ctrl+Shift+Space
Trigger parameter hints

Shift+F12
Show References

Ctrl+K Ctrl+X
Trim trailing whitespace

Ctrl+K M
Change file language

EDITOR

Ctrl+K Ctrl+ ←/→
Focus into previous/next editor group

Ctrl+K ← / →
Move active editor group

Ctrl+F4, Ctrl+W
Close editor

Ctrl+
Split editor

Ctrl+ 1 / 2 / 3
Focus into 1st, 2nd or 3rd editor group

Ctrl+K F
Close folder

Ctrl+Shift+PgUp / PgDn
Move editor left/right

Ctrl+M
Toggle Tab moves

Shift+Alt+A
Toggle block comment

FILE MANAGEMENT

Ctrl+N
New File

Ctrl+S
Save

Ctrl+Shift+S
Save As

Ctrl+O
Open File

Ctrl+K S
Save All

Ctrl+K Ctrl+W
Close All

Ctrl+Shift+T
Reopen closed editor

Ctrl+F4
Close

Ctrl+K
Enter Keep preview mode editor open

Ctrl+Shift+Tab
Open previous

Ctrl+K P
Copy path of active file

Ctrl+Tab
Open next

Ctrl+K R
Reveal active file in Explorer

Ctrl+K O
Show active file in new window/instance

DISPLAY

F11
Toggle full screen

Ctrl+Shift+E
Show Explorer / Toggle focus

Ctrl+Shift+G
Show Source Control

Ctrl+Shift+D
Show Debug

Ctrl+Shift+F
Show Search Display

Ctrl+Shift+X
Show Extensions

Ctrl+Shift+J
Toggle Search details

Ctrl+Shift+U
Show Output panel

Ctrl+Shift+H
Replace in files

Ctrl+Shift+V
Open Markdown preview

Ctrl+ +/ -
Zoom in/out

Ctrl+B
Toggle Sidebar visibility

Shift+Alt+0
Toggle editor layout (horizontal/vertical)

Ctrl+K Z
Zen Mode (Esc, Esc to exit)

Ctrl+K V
Open Markdown preview to the side

DEBUGGING

F5
Start/Continue Debug

F9
Toggle breakpoint

F10
Step over

Shift+F5
Stop

F11 / Shift+F11
Step into/out

Ctrl+K Ctrl+I
Show hover Integrated terminal

Ctrl+`
Show integrated terminal

Ctrl+C
Copy selection

Ctrl+V
Paste into active terminal

Ctrl+Shift+`
Create new terminal Integrated terminal

Ctrl+↑ / ↓
Scroll up/down

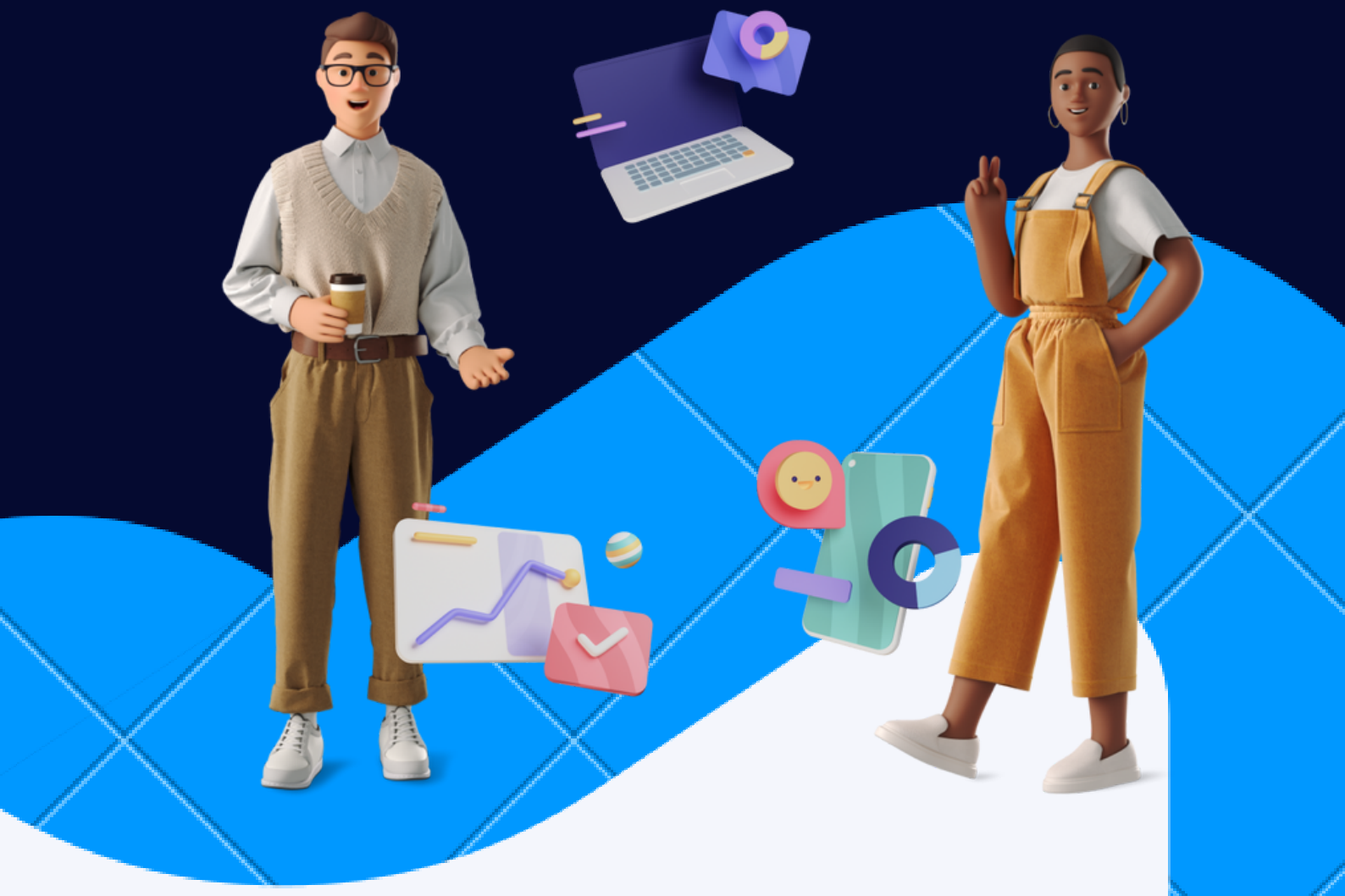
Ctrl+Home /
End Scroll to top/bottom

Shift+PgUp / PgDn
Scroll page up/down

CODE HELP

Git

GitHub CheatSheet



CODE HELP

GIT

Git is a version control system that is used for tracking changes in computer files and coordinating work on those files among multiple people. It is a distributed version control system, which means that it allows multiple users to work on the same files simultaneously and keeps track of changes made to the files by each user. Git is widely used in software development and has become a standard tool for collaborating on code.

This cheat sheet features the most important and commonly used Git commands for easy reference.

INSTALLATION & GUIS

With platform specific installers for Git, GitHub also provides the ease of staying up-to-date with the latest releases of the command line tool while providing a graphical user interface for day-to-day interaction, review, and repository synchronization.

GitHub for Windows: https://windows.github.com

GitHub for Mac: https://mac.github.com

For Linux and Solaris platforms, the latest release is available on the official Git web site.

Git for All Platforms: https://git-scm.com

SETUP

Configuring user information used across all local repositories.

git config --global user.name “[firstname lastname]”
set a name that is identifiable for credit when review version history.

git config --global user.email “[valid-email]”
set an email address that will be associated with each history marker.

git config --global color.ui auto
set automatic command line coloring for Git for easy reviewing.

SETUP & INIT

Configuring user information, initializing and cloning repositories.

git init
initialize an existing directory as a Git repository.

git clone [URL]
retrieve an entire repository from a hosted location via URL.

STAGE & SNAPSHOT

Working with snapshots and the Git staging area.

git status
show modified files in working directory, staged for your next commit.

git add [file]
add a file as it looks now to your next commit (stage).

git reset [file]
unstage a file while retaining the changes in working directory.

git diff
diff of what is changed but not staged.

git diff --staged
diff of what is staged but not yet committed.

git commit -m "[descriptive message]"
commit your staged content as a new commit snapshot.

BRANCH & MERGE

Isolating work in branches, changing context, and integrating changes.

git branch
list your branches, a* will appear next to the currently active branch.

git branch [branch-name]
create a new branch at the current commit.

git checkout
switch to another branch and check it out into your working directory.

git merge [branch]
merge the specified branch's history into the current one.

git log
show all commits in the current branch's history.

INSPECT & COMPARE

Examining logs, diffs and object information.

git log
show the commit history for the currently active branch.

git log branchB..branchA
show the commits on branchA that are not on branchB.

git log --follow [file]
show the commits that changed file, even across renames.

git diff branchB...branchA
show the diff of what is in branchA that is not in branchB.

git show [SHA]
show any object in Git in human-readable format.

TRACKING PATH CHANGES

Versioning file removes and path changes.

git rm [file]
delete the file from project and stage the removal for commit.

git mv [existing-path] [new-path]
change an existing file path and stage the move.

git log --stat -M
show all commit logs with indication of any paths that moved.

IGNORING PATTERNS

Preventing unintentional staging or committing of files.

logs/
***.notes**
pattern*/
Save a file with desired patterns as .gitignore with either direct string matches or wildcard globs.

git config --global core.excludesfile [file]
system wide ignore pattern for all local repositories.

SHARE & UPDATE

Retrieving updates from another repository and updating local repository.

git remote add [alias] [URL]
add a git URL as an alias.

git fetch [alias]
fetch down all the branches from that Git remote.

git merge [alias]/[branch]
merge a remote branch into your current branch to bring it up to date.

git push [alias] [branch]
Transmit local branch commits to the remote repository branch.

git pull
fetch and merge any commits from the tracking remote branch.

REWRITE HISTORY

Rewriting branches, updating commits and clearing history.

git rebase [branch]
apply any commits of current branch ahead of specified one.

git reset --hard [commit]
clear staging area, rewrite working tree from specified commit.

TEMPORARY COMMITS

Temporarily store modified, tracked files in order to change branches.

git stash
Save modified and staged changes.

git stash list
list stack-order of stashed file changes.

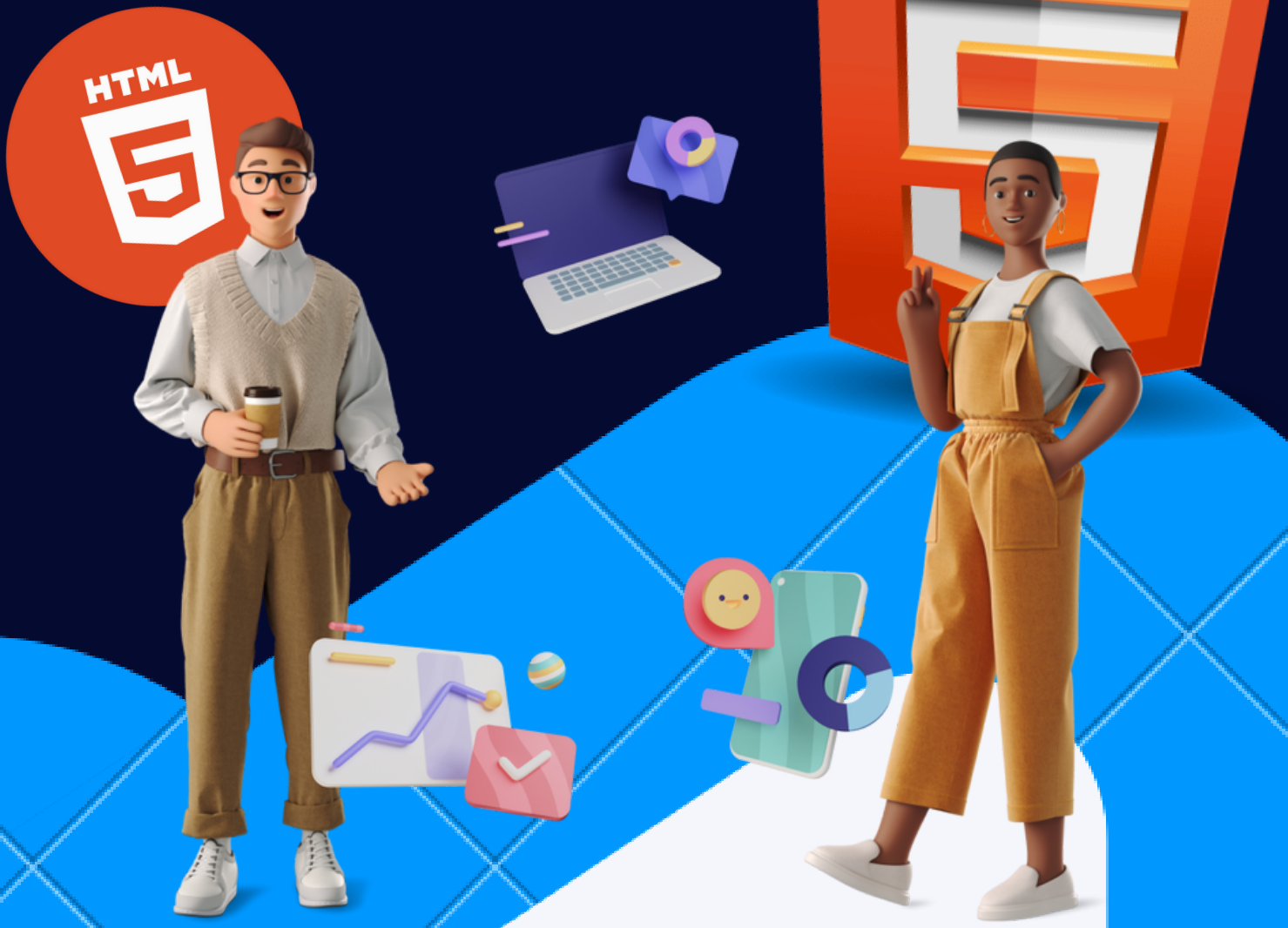
git stash pop
write working from top of stash stack.

git stash drop
discard the changes from top of stash stack.

CODE HELP

HTML

CheatSheet



CODE HELP

In this Cheatsheet, we will cover the basics of HTML tags, elements, and attributes. We will provide examples to help you understand how these elements work and how to use them in your own web development projects. Whether you are a beginner or an experienced developer, this PDF can serve as a useful reference guide.

HTML

HTML (Hypertext Markup Language) is a standard markup language used to create web pages. It is used to structure and format content on the web, including text, images, and other multimedia elements. HTML consists of a series of elements that are represented by tags, which are used to define the structure and content of a webpage.
HTML is an essential part of the web development process and is used to create the structure and content of websites. It is a fundamental skill for web developers and is used to create the majority of websites on the internet.

HTML COMPONENTS

- **<html> tag:** This tag acts as a container for every other element in the document except the <!DOCTYPE html> tag.
- **<head> tag:** Includes all the document's metadata.
- **<title> tag:** Defines the title of the document which is displayed in the browser's title bar.
- **<body> tag:** Acts as a container for the document's content that gets displayed on the browser.

This is how it all comes together:

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <title> Code Help HTML Cheat Sheet </title>
    </head>
    <body> .... </body>
</html>
```

<!DOCTYPE html> specifies that we are working with an HTML5 document.

The following tags contribute extra information to the HTML document:

- **<meta> tag**: This tag can be used to define additional information about the webpage.
- **<link> tag:** Used to link the document to an external resource.
- **<style> tag:** Used for defining styles for the document.
- **<script> tag:** Used to write code snippets (usually JavaScript) or to link the document to an external script.

```
<head>
  <meta charset="UTF-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=edge" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <link rel="stylesheet" href="style.css" />
  <title>Code Help HTML Cheat Sheet</title>

  <style>
   *{
     font-size: 40px;
   }
  </style>

  <script>
        alert ('message');
  </script>

</head>
```

STRUCTURE OF A HTML DOCUMENT

While constructing your HTML document, you can use certain tags to establish its structure. The **<h1>** to **<h5>** tags signify different heading levels, with **<h1>** being the highest level and **<h5>** being the lowest level.

```
<h1> Heading 1 </h1>
<h2> Heading 2 </h2>
<h3> Heading 3 </h3>
<h4> Heading 4 </h4>
<h5> Heading 5 </h5>
```

You use the **<p>** tag to create a paragraph.

```
<p> This is a paragraph </p>
```

The ** ** tag can be employed to segment and style different areas of the document. It also acts as a parent container for other elements within the document.

This is how it works:

```
<div class="About Us">
    <h1> This is the About Us section </h1>
    <p> Welcome to the About Us section! </p>
</div>

<div class="Contact Us">
    <h1> This is the Contact Us section </h1>
    <p> Contact us on 0123456789 </p>
</div>
```

We also have the **** tag. This is similar to ** ** but you use it as an inline container.

```
<p> Hello <span class="span1"> World! </span></p>
```

There's the **
** tag, which we use to insert line breaks in the document. This tag does not require a closing tag.

```
<p> Welcome to <br/> Code Help </p>
```

The **<hr/>** tag is used to create a horizontal line. It also has no closing tag.

```
<p> Welcome to <hr/> Code Help </p>
```

IMAGES IN HTML

In HTML, we use the **<img/>** tag to insert images into the document.

Here are some attributes of the **<img/>** tag.

- **src** is used to specify the location of the image on your computer or the internet.
- **alt** specifies alternative text that displays if the image cannot be rendered. This text is also helpful for screen readers.
- **height** determines the height of the image.
- **width** determines the width of the image.

- **border** specifies the thickness of the borders around the image. If no border is added, it is set to 0.

```
<img src="logo.png" alt="Code Help Logo" width="200" height="200">
```

TEXT FORMATING IN HTML

HTML provides multiple methods for formatting text. Let's take a brief look at them now:

- The **<i>** tag formats text in italics.
- The **<b>** tag formats text in bold.
- The **** tag also formats text in bold and is used to emphasize important information.
- The **** tag is another emphasis tag that formats text in italics.
- The **<sub>** tag formats text as subscript, like Carbon Dioxide CO_2.
- The **<sup>** tag formats text as a superscript, like the power of a number, 2^{32}.
- The **<small>** tag decreases the size of text.
- The **<del>** tag formats text as deleted by striking a line through it.
- The **<address>** tag is used to show the author's contact information.
- The **<abbr>** tag denotes an abbreviation.
- The **<code>** tag formats text as code snippets.
- The **<mark>** tag highlights text.
- The **<ins>** tag formats text as inserted, which is usually underlined.

- The **<blockquote>** tag is used to enclose a section of text quoted from another source.
- The **<q>** tag is used for shorter inline quotes.
- The **<cite>** tag is used to cite the author of a quote.

```
<p><i> Italic </i></p>
<p><b> Bold </b></p>
<p><strong> Strong </strong></p>
<p><em> Strong </em></p>
<p><sub> Subscript </sub></p>
<p><sup> Superscript </sup></p>
<p><small> Small </small></p>
<p><del> Delete </del></p>
<p><address> Address </address></p>
<p><abbr> Inserted Abbreviation </abbr></p>
<p><code> Code Snippet </code></p>
<p><mark> Marked Text </mark></p>
<p><ins> Insert </ins></p>
<p><blockquote> Quoted </blockquote></p>
<p><q> Short Quoted </q></p>
<p><cite> Cited </cite></p>
```

LINKS IN HTML

The **<a>** tag, also referred to as the anchor tag, is used to establish hyperlinks that link to other pages (external web pages included) or to a particular section within the same page.

Here are some attributes of the **<a>** tag:

- The **href** attribute specifies the URL that the link will take the user to when clicked.
- The **download** attribute specifies that the target or resource clicked is a downloadable file.
- The **target** attribute specifies where the linked document or resource should be opened. This could be in the same window or a new window.

```
<a href="https://www.thecodehelp.in/" target="_blank"> Code Help </a>
```

LISTS IN HTML

- The **<ol>** tag defines an ordered list.
- The **<ul>** tag defines an unordered list.
- The **<li>** tag is used to create items in the list.

```
<!-- Unordered List -->
<ul>
    <li> Course 1</li>
    <li> Course 2 </li>
    <li> Course 3</li>
</ul>

<!-- Ordered List -->
<ol>
    <li> Course 1 </li>
   <li> Course 2 </li>
    <li> Course 3 </li>
</ol>
```

FORMS IN HTML

The **<form>** element is used to create a form in HTML. Forms are used to gather user input.

Some attributes associated with the **<form>** element include:

- The **action** attribute specifies where the form data should be sent when the form is submitted.
- The **target** attribute specifies where to display the form's response.
- The **autocomplete** attribute can have a value of on or off and determines whether the browser should automatically fill in the form.
- The **novalidate** attribute specifies that the form should not be validated.
- The **method** attribute specifies the HTTP method to use when sending form data.
- The **name** attribute specifies the name of the form.
- The **required** attribute specifies that an input element cannot be left blank.
- The **autofocus** attribute gives focus to the input elements when the page loads.
- The **disabled** attribute disables an input element, preventing the user from interacting with it.
- The **placeholder** attribute is used to provide a hint to the user about what information is required for the input element.

Other input elements that can be used in forms include:

- **<textarea>:** allows users to enter multiple lines of text as input.
- **<select>:** provides a list of options for users to choose from.
- **<option>:** creates a single option within a **<select>** element.
- **<input>:** provides an input field for users to enter data. The **type** attribute specifies the type of data that can be entered.
- **<button>:** creates a button that can be clicked to perform an action.

```
<form action="/Submit_URL/" method="post">
  <label for="FirstName"> First Name: </label>
    <input type="text"
           name="FirstName"
           placeholder="First Name"
           required >

   <label for="LastName"> Last Name: </label>
    <input type="text"
           name="LastName"
           placeholder="Last Name"
           required >
  <label for="add"> Address: </label>
  <textarea name="add"></textarea>
  <label for="age"> Age: </label>
  <select id="age">
    <option value="11-20">11-20</option>
    <option value="21-30">21-30</option>
    <option value="31-40">31-40</option>
    <option value="41-40">41-50</option>
  </select>

 <input type="submit" value="Submit">
</form>
```

TABLES IN HTML

The **<table>** tag defines a HTML table.

- **<thead>:** defines the header information for each column in the table.
- **<tbody>:** defines the body or content of the table.
- **<tfoot>:** defines the footer information of the table. <tr>: represents a row in the table.
- **<td>:** represents a single cell in the table.
- **<th>:** represents the heading for a column of values in the table.

```
<table>
    <thead>
        <tr>
            <th> Name </th>
            <th> CGPA </th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td> Koushik Sadhu </td>
            <td> 9.66 </td>
        </tr>
        <tr>
            <td> Pranay Gupta </td>
            <td> 9.72 </td>
        </tr>
    </tbody>
    <tfoot>
        <tr>
            <td> Nidhi Gupta </td>
            <td> 10 </td>
        </tr>
    </tfoot>
</table>
```

TAGS IN HTML

The following tags were introduced in HTML5:

- **<header> tag:** defines the header section of a webpage.
- **<footer> tag:** defines the footer section of a webpage.
- **<main> tag:** defines the main content section of a webpage.
- **<article> tag:** defines a standalone section of content, such as an article.
- **<nav> tag:** used to contain navigation links.
- **<meter> tag:** used to measure data within a given range.
- **<progress> tag:** used as a progress bar to indicate the completion of a task.
- **<dialog> tag:** used to create a dialog box.
- **<audio> tag:** used to embed an audio file on a webpage.
- **<video> tag:** used to embed a video on a webpage.
- **<section> tag:** defines a section within a webpage.
- **<aside> tag:** often used for content placed in a sidebar.
- **<time> tag:** used for formatting dates and times.
- **<figure> tag:** used for figures such as charts.
- **<figcaption> tag:** provides a description for a **<figure>**.

```
<header>
      <h1> Welcome to CodeHelp! </h1>
</header>

<nav>
   <ul>
         <li><a href="#">About Us</a></li>
         <li><a href="#">Courses</a></li>
         <li><a href="#">Contact</a></li>
   </ul>
</nav>

<article>
      <h1> About CodeHelp </h1>
      <p> This is all about CodeHelp. </p>
      <aside>
            <p> Book your seat now. </p>
      </aside>
</article>

<progress min="0" max="100" value="10"> </progress>

<footer> Copyright © 2023 CodeHelp. All Rights Reserved. </footer>
```

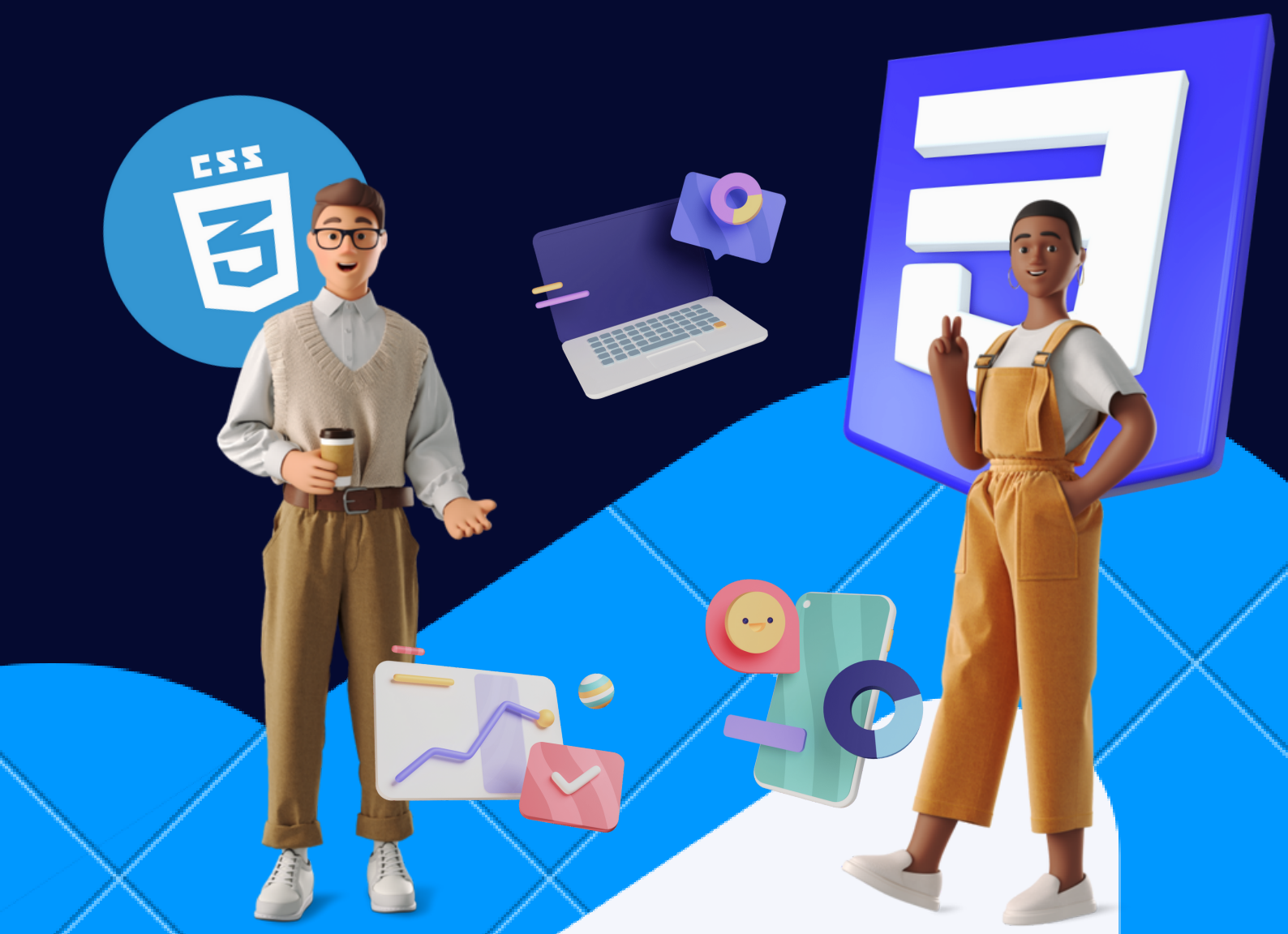
CHARACTER AND SYMBOLS

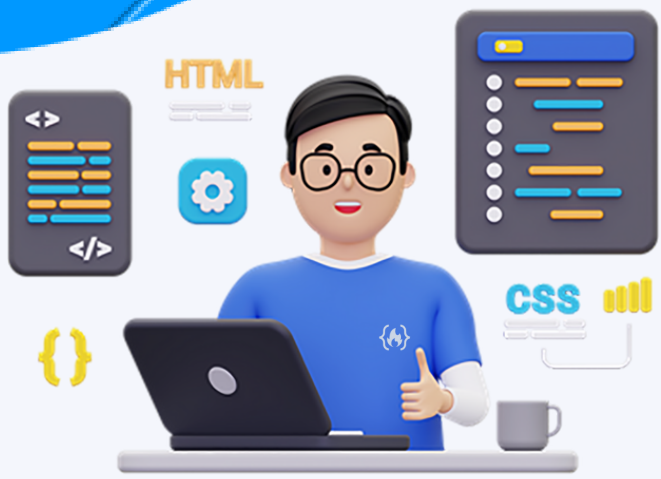
In HTML documents, some symbols may not be directly available on the keyboard. However, there are several ways to include these symbols in a document. These include using the symbol's entity name, decimal value, or hexadecimal value.

- Copyright Symbol: **©**
- Dollar Symbol: **$**
- Ampersand Symbol: **&**
- Greater than Symbol: **>**
- Less than Symbol: **<**

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <title> Code Help HTML Cheat Sheet </title>
    </head>
        <body>
                <p> Copyright Symbol: &copy; </p>
                <p> Dollar Symbol: &dollar; </p>
                <p> Ampersand Symbol: &amp; </p>
                <p> Greater than Symbol: &gt; </p>
                <p> Less than Symbol: &lt; </p>
        </body>
</html>
```

CSS CheatSheet





CODE HELP

In this Cheatsheet, we will cover the basics of CSS properties. We will provide examples to help you understand how these properties work and how to use them in your own web development projects. Whether you are a beginner or an experienced developer, this PDF can serve as a useful reference guide.

CSS

CSS (**Cascading Style Sheets**) is a stylesheet language used for describing the look and formatting of a document written in HTML (Hypertext Markup Language).

CSS is used to define the layout, font, color, and other visual aspects of a webpage, and it helps to separate the content of a webpage from its presentation. CSS allows you to apply styles to multiple HTML elements at once, and it makes it easy to maintain and update the styling of a webpage.

You can use CSS to specify styles for different devices, such as desktop computers, tablets, and mobile phones, which makes it easier to create responsive designs that look good on any device. To use CSS, you can include a stylesheet in your HTML document using a <link> element, or you can use inline styles in your HTML elements using the style attribute.

You can also use CSS to style elements in other documents, such as XML or SVG, and you can use CSS in combination with other technologies, such as JavaScript, to create dynamic and interactive webpages.

FONT PROPERTIES IN CSS

The font has many properties that you can change, such as its face, weight, and style, which allow you to alter the appearance of your text.

- **Font-Family:** Specifies the font family of the text.
- **Font:** A shorthand property for specifying the font-style, font-variant, font-weight, font-size/line-height, and font-family properties all at once.
- **Font-Size:** Specifies the font size of the text.
- **Font-Weight:** Specifies the weight of a font.
- **Font-Style:** Specifies the font style of the text.
- **Font-Variant:** Specifies whether or not the text is displayed in a small-caps font.

```
p {
    font-family: Times, serif, Arial, Helvetica, sans-serif;
    font: 15px Helvetica, sans-serif, Arial;
    font-size: 15px;
    font-weight: bold;
    font-style: italic;
    font-variant: small-caps;
}
```

TEXT PROPERTIES IN CSS

CSS has a lot of properties for formatting text.

- **Text-Align:** It specifies the horizontal alignment of text.
- **Text-Decoration:** It specifies the decoration added to text.
- **Letter-Spacing:** It increases or decreases the space between characters in a text.
- **Text-Transform:** It controls the capitalization of text.
- **Word-Spacing:** It increases or decreases the space between words in a text.
- **Text-Indent:** It specifies the indentation of the first line in a text-block.
- **Line-Height:** It sets the line height.
- **Text-Shadow:** It adds shadow to the text.

```
p {
    text-align: center;
    text-decoration: underline;
    letter-spacing: 5px;
    text-transform: uppercase;
    word-spacing: 8px;
    text-indent: 40px;
    line-height: 40%;
    text-shadow: 4px 4px #ff0000;
}
```

BACKGROUND PROPERTIES IN CSS

The background properties allow you to customize the color, image, position, size, and other aspects of the document's background. As the name implies, these properties affect the background of the document.

- **Background-Image:** It specifies one or more background images for an element.
- **Background-Size:** It specifies the size of the background images.
- **Background-Position:** It specifies the position of a background image.
- **Background-Repeat:** It sets, how a background image will be repeated.
- **Background-Color:** It specifies the background color of an element.
- **Background-Attachment:** It sets whether a background image scrolls with the rest of the page, or is fixed.
- **Background-Origin:** It specifies the origin position of a background image.
- **Background:** A shorthand property for all the background-* properties.

```
body {
      background-image:  url('codeHelp.png');
      background-size:  auto;
      background -position:  center;
      background -repeat:  no-repeat;
      background -color:  #ffffff;
      background -attachment:  fixed;
      background -origin:  content-box;
      background :  url('codeHelp.png');
}
```

BORDERS PROPERTIES IN CSS

The border properties enable you to alter the style, radius, color, and other characteristics of the buttons or other elements within the document.

- **Border-Width:** It sets the width of the four borders.
- **Border-Style:** It sets the style of the four borders.
- **Border-Radius:** A shorthand property for the four border-*-radius properties.
- **Border-Color:** It sets the color of the four borders.

- **Border:** A shorthand property for border-width, border-style and border-color.

- **Border-Spacing:** It sets the distance between the borders of adjacent cells.

```
div {
    border-width: 4px;
    border-style: solid;
    border-radius: 4px;
    border-color: #000000;
    border: 20px dotted coral;
    border-spacing: 20px;
}
```

BOX MODEL PROPERTIES IN CSS

The CSS box model is a structure that encloses every HTML element, and is composed of margins, borders, padding, and the element's content. This model is utilized to design and arrange the layout of web pages.

- **Padding:** A shorthand property for all the padding-* properties.

- **Visibility:** It specifies whether or not an element is visible.

- **Display:** It specifies how a certain HTML element should be displayed.

- **Height:** It sets the height of an element.
- **Width:** It sets the width of an element.
- **Float:** It specifies whether an element should float to the left, right, or not at all.
- **Clear:** It specifies what should happen with the element that is next to a floating element.
- **Margin:** It sets all the margin properties in one declaration.
- **Overflow:** It specifies what happens if content overflows an element's box.

```
p {
    padding: 10px 20px 10px 20px;
    visibility: hidden;
    display: inline-block;
    height: auto;
    width: 100px;
    float: right;
    clear: left;
    margin: 20px 10px 20px 10px;
    overflow: scroll;
}
```

COLORS PROPERTIES IN CSS

The color property can be used to add color to various objects.

- **Color:** It sets the color of text.
- **Outline-Color:** It sets the color of an outline.
- **Caret-Color:** It specifies the color of the cursor (caret) in inputs, textareas, or any element that is editable.
- **Opacity:** It sets the opacity level for an element.

```
{
    color: rgb(0, 0, 0);
    outline-color: #000000;
    caret-color: coral;
    opacity: 0.8;
}
```

LAYOUT PROPERTIES IN CSS

It defines the appearance of the content within a template.

- **Box-Align:** It specifies how an element aligns its contents across its layout in a perpendicular direction.
- **Box-Direction:** It specifies whether a box lays out its contents normally (from the top or left edge), or in reverse (from the bottom or right edge).
- **Box-Flex:** This property specifies how a box grows to fill the box that contains it, in the direction of the containing box's layout.
- **Box-Orient:** It sets whether an element lays out its contents horizontally or vertically.
- **Box-Sizing:** It allows us to include the padding and border in an element's total width and height.
- **Box-Pack:** This property specifies how a box packs its contents in the direction of its layout.
- **Min-Width:** It sets the minimum width of an element.
- **Max-Width:** It sets the maximum width of an element.
- **Min-Height:** It sets the minimum height of an element.
- **Max-Height:** It sets the maximum height of an element.

```
{
    box-align:  start;
    box-direction:  normal;
    box-flex:  normal;
    box-orient:  inline;
    box-sizing:  margin-box;
    box-pack:  justify;
    min-width:  200px;
    max-width:  400px;
    min-height: 100px;
    max-height:  1000px;
}
```

TABLE PROPERTIES IN CSS

Table properties allow you to customize the appearance of tables in a document, including options like border spacing, table layout, and captions.

- **Border-Spacing:** It sets the distance between the borders of adjacent cells.
- **Border-Collapse:** It sets whether table borders should collapse into a single border or be separated.
- **Empty-Cells:** It specifies whether or not to display borders and background on empty cells in a table.
- **Caption-Side:** It specifies the placement of a table caption.

- **Table-Layout:** It defines the algorithm used to layout table cells, rows, and columns.

```
{
    border-spacing:  4px;
    border-collapse:  separate;
    empty-cells:  show;
    caption-side:  bottom;
    table-layout:  auto;
}
```

COLUMNS PROPERTIES IN CSS

These properties are specifically applied to the columns of a table and are used to enhance its appearance.

- **Column-Gap:** It specifies the gap between the columns.
- **Column-Rule-Width:** It specifies the width of the rule between columns.
- **Column-Rule-Color:** It specifies the color of the rule between columns.
- **Column-Rule-Style:** It Specifies the style of the rule between columns.
- **Column-Count:** It specifies the number of columns an element should be divided into.
- **Column-Span:** It specifies how many columns an element should span across.

- **Column-Width:** It specifies the column width.

```
{
    column-gap:  4px;
    column-rule-width:  medium;
    column-rule-color:  #000000;
    column-rule-style:  dashed;
    column-count:  20;
    column-span:  all;
    column-width:  4px;
}
```

LIST & MARKER PROPERTIES IN CSS

The list and marker properties can be used to customize the appearance of lists in a document.

- **List-Style-Image:** It specifies an image as the list-item marker.
- **List-Style-Position:** It specifies the position of the list-item markers (bullet points).
- **List-Style-Type:** It specifies the type of list-item marker.
- **Marker-Offset:** It allows you to specify the distance between the marker and the text relating to that marker.

```
{
    list-style-image:  url('codeHelp.png');
    list-style-position:  10px;
    list-style-type:  square;
    marker-offset:  auto;
}
```

ANIMATION PROPERTIES IN CSS

CSS animations enable the creation of animated transitions or the animation of other media elements on a web page.

- **Animation-Name:** It specifies a name for the @keyframes animation.
- **Animation-Delay:** It specifies a delay for the start of an animation.
- **Animation-Duration:** It specifies a delay for the start of an animation.
- **Animation-Timing-Function:** It specifies the speed curve of an animation.
- **Animation-Iteration-Count:** It specifies the number of times an animation should be played.
- **Animation-Fill-Mode:** It specifies a style for the element when the animation is not playing (before it starts, after it ends, or both).

- **Animation-Play-State:** It specifies whether the animation is running or paused.

- **Animation-Direction:** It specifies whether the animation is running or paused.

```
{
    animation-name: anime;
    animation-delay: 4ms;
    animation-duration: 10s;
    animation-timing-function: ease;
    animation-iteration-count: 5;
    animation-fill-mode: both;
    animation-play-state: running;
    animation-direction: normal;
}
```

TRANSITION PROPERTIES IN CSS

Transitions allow you to specify how an element will change from one state to another.

- **Transition-Property:** It specifies the name of the CSS property the transition effect is for.

- **Transition-Delay:** It specifies when the transition effect will start.

- **Transition-Duration:** It specifies how many seconds or milliseconds a transition effect takes to complete.

- **Transition-Timing-Function:** It specifies the speed curve of the transition effect.

```
{
    transition-property:  none;
    transition-delay:   4ms;
    transition-duration:   10s;
    transition-timing-function:   ease-in-out;
}
```

CSS FLEXBOX

Flexbox is a CSS layout system that makes it easy to align and distribute items within a container using rows and columns. It allows items to "flex" and adjust their size to fit the available space, making responsive design simpler to implement. Flexbox makes formatting HTML elements more straightforward and efficient.

PARENT PROPERTIES:

- **Display:** It specifies how a certain HTML element should be displayed.
- **Flex-Direction:** It specifies the direction of the flexible items.
- **Flex-Wrap:** It specifies whether the flexible items should wrap or not.
- **Flex-Flow:** It is a shorthand property for the flex-direction and the flex-wrap properties.

- **Justify-Content:** It specifies the alignment between the items inside a flexible container when the items do not use all available space.

- **Align-Items:** It specifies the alignment for items inside a flexible container.

- **Align-Content:** It specifies the alignment between the lines inside a flexible container when the items do not use all available space.

```
{
    display: flex;
    flex-direction: row | row-reverse | column | column-reverse;
    flex-wrap: nowrap | wrap | wrap-reverse;
    flex-flow: column wrap;
    justify-content: flex-start | flex-end | center | space-between | space-around | space-evenly | start | end | left | right ... + safe | unsafe;
    align-items: stretch | flex-start | flex-end | center | baseline | first baseline | last baseline | start | end | self-start | self-end + ... safe | unsafe;
    align-content: flex-start | flex-end | center | space-between | space-around | space-evenly | stretch | start | end | baseline | first baseline | last baseline + ... safe | unsafe;
}
```

CHILD PROPERTIES:

- **Order:** It sets the order of the flexible item, relative to the rest.
- **Flex-Grow:** It specifies how much the item will grow relative to the rest.
- **Flex-Shrink:** It specifies how the item will shrink relative to the rest.
- **Flex-Basis:** It Specifies the initial length of a flexible item.
- **Align-Self:** It specifies the initial length of a flexible item.

```
{
    order:  2;                          /* By default it is 0 */
    flex-grow:  5;                      /* By default it is 0 */
    flex-shrink:  4;                    /* By default it is 1 */
    flex-basis: | auto;                 /* By default it is auto */
    align-self:  auto | flex-start | flex-end | center | baseline | stretch;
    flex:  none | [ <'flex-grow'> <'flex-shrink'>? || <'flex-basis'> ];
}
```

CSS GRID

The Grid layout is a 2-dimensional system in CSS that allows for the creation of complex and responsive web design layouts with consistent results across different browsers. It makes it easier to build these types of layouts.

PARENT PROPERTIES:

- **Display:** It specifies how a certain HTML element should be displayed.
- **Grid-Template-Columns:** It specifies the size of the columns, and how many columns in a grid layout.
- **Grid-Template-Rows:** It specifies the size of the columns, and how many columns in a grid layout.
- **Grid-Template:** It is a shorthand property for the grid-template-rows, grid-template-columns and grid-areas properties.
- **Column-Gap:** It specifies the gap between the columns.
- **Row-Gap:** It specifies the gap between the grid rows.
- **Grid-Column-Gap:** It specifies the size of the gap between columns.
- **Grid-Row-Gap:** It specifies the size of the gap between rows.
- **Gap:** It is a shorthand property for the row-gap and the column-gap properties.

- **Grid-Gap:** It is a shorthand property for the grid-row-gap and grid-column-gap properties.

- **Align-Items:** It specifies the alignment for items inside a flexible container.

- **Justify-Content:** It specifies the alignment between the items inside a flexible container when the items do not use all available space.

- **Align-Content:** It specifies the alignment between the lines inside a flexible container when the items do not use all available space.

- **Grid-Auto-Columns:** It specifies a default column size.

- **Grid-Auto-Rows:** It specifies a default row size.

- **Grid-Auto-Flow:** It specifies how auto-placed items are inserted in the grid.

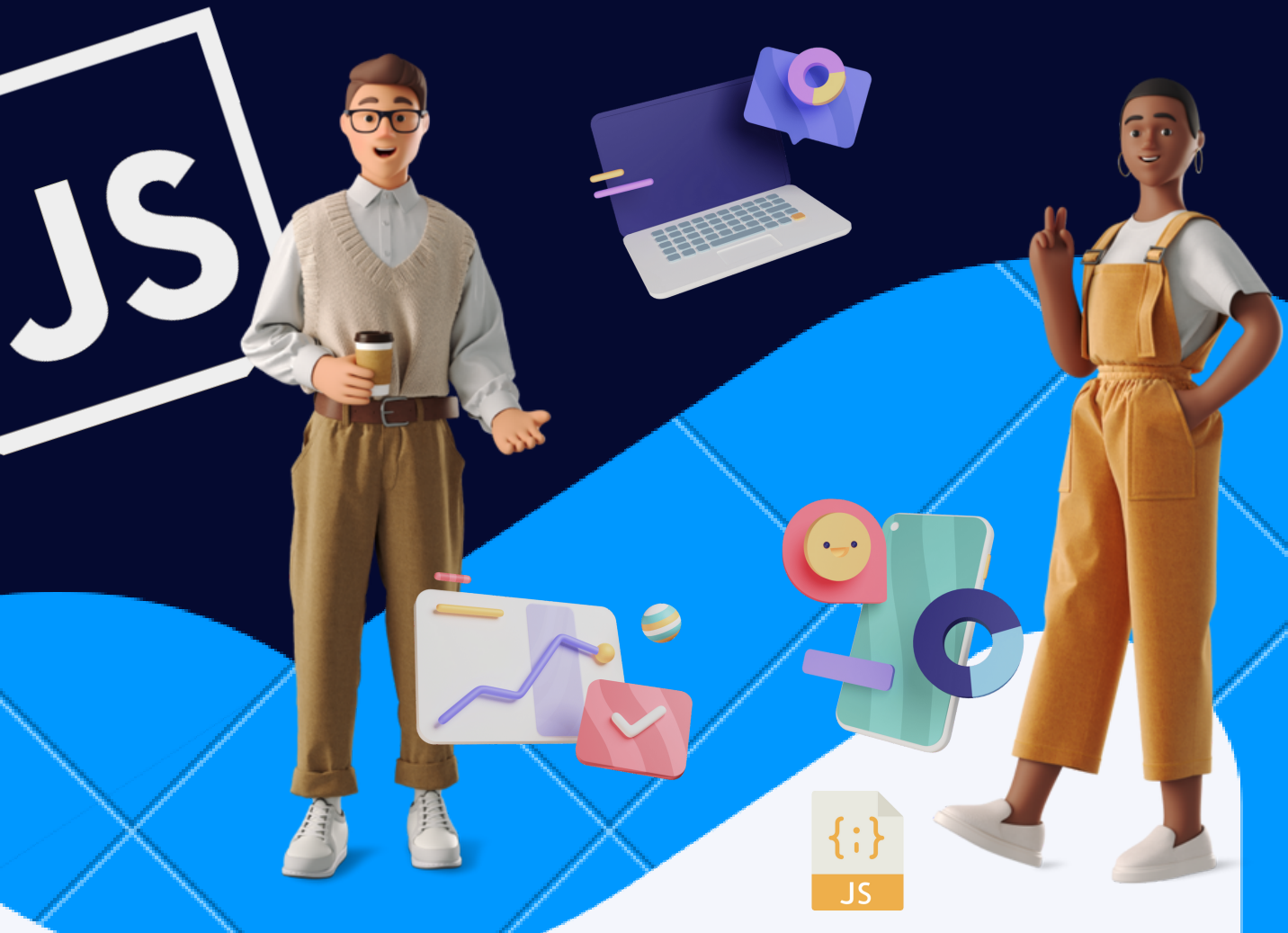
```
{
    display:  grid | inline-grid;
    grid-template-columns:  10px 10px 10px;
    grid-template-rows:  5px auto 10px;
    grid-template:  none | <grid-template-rows> / <grid-template-
    columns>;
    column-gap:  <line-size>;
    row-gap:  <line-size>;
    grid-column-gap:  <line-size>;
    grid-row-gap:  <line-size>;
    gap:  <grid-row-gap> <grid-column-gap>;
    grid-gap:  <grid-row-gap> <grid-column-gap>;
    align-items:  start | end | center | stretch;
    justify-content:  start | end | center | stretch | space-around | space-
    between | space-evenly;
    align-content:  start | end | center | stretch | space-around | space-
    between | space-evenly;
    grid-auto-columns:  <track-size>;
    grid-auto-rows:  <track-size>;
    grid-auto-flow:  row | column | row dense | column dense;
}
```

CHILD PROPERTIES:

- **Grid-Column-Start:** It specifies where to start the grid item.
- **Grid-Column-End:** It specifies where to end the grid item.
- **Grid-Row-Start:** It specifies where to start the grid item.
- **Grid-Row-End:** It specifies where to end the grid item.
- **Grid-Column:** It is a shorthand property for the grid-column-start and the grid-column-end properties.
- **Grid-Row:** It is a shorthand property for the grid-row-start and the grid-row-end properties.
- **Grid-Area:** It either specifies a name for the grid item, or this property is a shorthand property for the grid-row-start, grid-column-start, grid-row-end, and grid-column-end properties.
- **Align-Self:** It specifies the alignment for selected items inside a flexible container.

```
{
    grid-column-start:  <number> | <name> | span <number> | span <name>
    | auto;
    grid-column-end:  <number> | <name> | span <number> | span <name> |
    auto;
    grid-row-start:  <number> | <name> | span <number> | span <name> |
    auto;
    grid-row-end:  <number> | <name> | span <number> | span <name> |
    auto;
    grid-column:  <start-line> / <end-line> | <start-line> / span <value>;
    grid-row:  <start-line> / <end-line> | <start-line> / span <value>;
    grid-area:  <name> | <row-start> / <column-start> / <row-end> /
    <column-end>;
    align-self:  start | end | center | stretch;
}
```

JavaScript CheatSheet



CODE HELP

In this Cheatsheet, we will cover the basics of JavaScript. We will provide examples to help you understand how JavaScript work and how to use them in your own web development projects. Whether you are a beginner or an experienced developer, this PDF can serve as a useful reference guide.

JavaScript

JavaScript is a programming language that is widely used in web development. It is a client-side scripting language, which means that it is executed on the client side (in a web browser) rather than on the server side. JavaScript is used to create interactive web pages and is an essential component of modern web development. It is a high-level, dynamically typed language that is interpreted, which means that it is not compiled into machine code before it is run. JavaScript is used to add functionality to websites, such as handling user events (such as clicks and form submissions), animating page elements, and making asynchronous requests to servers. It is a versatile language that can be used for a wide range of applications, from simple scripts that add simple interactivity to websites to complex single-page applications.

On page script:

```
<script type="text/javascript"> .... </script>
```

Include external JS file:

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

Delay - 1 second timeout:

```
setTimeout(function () {


}, 1000);
```

Functions:

```
function addNumbers(a, b) {
return a + b; ;
}
x = addNumbers(1, 2);
```

Edit DOM element:

```
document.getElementById("elementID").innerHTML = "Hello World!";
```

Output:

```
console.log(a);          // write to the browser console.
document.write(a);       // write to the HTML.
alert(a);                // output in an alert box.
confirm("?");            // yes/no dialog, returns true/false depending on user click.
prompt("Age ?","0");     // input dialog box. (Initial Value = 0).
```

Comments:

```
/* Multi line comment */
// One line
```

Variables:

```
var v;                          // variable
var b = "CodeHelp";             // string
var c = "Code" + " " + "Help";  // = "Code Help"
var d = 1 + 4 + "5";            // = "55"
var e = [1,2,6,8];              // array
var f = true;                   // boolean
var g = /()/;                   // RegEx
var h = function(){};           // function object
const PI = 3.14;                // constant
var a = 7, b = 6, c = a + b;    // one line
let a = 'aaa';                  // block scope local variable
```

Strict mode:

```
"use strict";          // Use strict mode to write secure code
x = 1;                 // Throws an error because variable is not declared
```

Values:

```
false, true                    // Boolean
4, 3.14, 0b10011, 0xF6, NaN    // Number
"CodeHelp", 'Love'             // String
undefined, Infinity, null      // Special
```

Basic Operators:

```
a = b + c - d;          // Addition, Subtraction.
a = b * (c / d);        // Multiplication, Division.
x = 10 % 4;             // Modulo, 10 / 4 Remainder = 2.
a++; b--;               // Postfix Increment and Decrement.
```

Bitwise Operators:

&	AND	5 & 1	(0101 & 0001)	1	(1)
|	OR	5 | 1	(0101 | 0001)	5	(101)
~	NOT	~ 5	(~0101)	10	(1010)
^	XOR	5 ^ 1	(0101 ^ 0001)	4	(100)
<<	Left Shift	5 << 1	(0101 << 1)	10	(1010)
>>	Right Shift	5 >> 1	(0101 >> 1)	2	(10)
>>>	Zero Fill Right Shift	5 >>> 1	(0101 >>> 1)	2	(10)

Arithmetic Operators:

```
x * (y + z)          // grouping
x = 4                // assignment
x == y               // equals
x != y               // unequal
x === y              // strict equal
x !== y              // strict unequal
x < y   x > y        // less and greater than
x <= y  x >= y       // less or equal, greater or equal
x += y               // x = x + y
x && y               // logical AND
x || y               // logical OR
!(x == y)            // logical NOT
x != y               // not equal
typeof x             // type of x
x << 2  x >> 3       // shifting
```

Objects:

```
var student = {                                  // object name
        firstName: "Koushik",                    // list of properties and values
        lastName: "Sadhu",
        age: 20,
        height: 175,
        fullName: function() {                   // object function
                return this.firstName + " " + this.lastName;
        }
};
student.age = 19;                                // setting value
student[age]++;                                  // incrementing value
name = student.fullName();                       // call object function
```

Strings:

```
var a = "Codehelp";
var b = 'I don\'t \n know';      // \n new line.
var len = a.length;              // string length.
a.indexOf("h");                  // find substring, -1 if doesn't contain.
a.lastIndexOf("e");              // last occurrence.
a.slice(3, 6);                   // cut out "ehe", negative values count from behind.
a.replace("help","love");        // find and replace, takes regular expressions.
a.toUpperCase();                 // convert to upper case.
a.toLowerCase();                 // convert to lower case.
a.concat(" ", str2);             // Codehelp + " " + str2.
a.charAt(4);                     // character at index 4: "h".
abc.split(" ");                  // splitting a string on space, and stores in an array.
```

Numbers and Math:

```
var pi = 3.14;
pi.toFixed(0);                  // returns 3
pi.toFixed(2);                  // returns 3.14, working with money
pi.toPrecision(2)               // returns 3.1
pi.valueOf();                   // returns number
Number(true);                   // converts to number
Number(new Date())              // number of milliseconds since, 1970
parseInt("3 months");           // returns the first number 3
parseFloat("3.5 days");         // returns 3.5
Number.MAX_VALUE                // largest possible JS number
Number.MIN_VALUE                // smallest possible JS number
Number.NEGATIVE_INFINITY        // Minus Infinity
Number.POSITIVE_INFINITY        // Infinity
var pi = Math.PI;               // 3.141592653589793
Math.round(4.5);                // 5
Math.pow(2,8);                  // 256 - 2 to the power of 8
Math.sqrt(49);                  // 7 - square root
Math.abs(-3.14);                // 3.14 - absolute, positive value
Math.ceil(3.14);                // 4 - rounded up
Math.floor(3.99);               // 3 - rounded down
Math.sin(0);                    // 0 - sine
Math.cos(Math.PI);              // OTHERS: tan, atan, asin, acos,
Math.min(0, 3, -2, 2);          // -2 the lowest value
Math.max(0, 3, -2, 2);          // 3 the highest value
Math.log(1);                    // 0 natural logarithm
Math.exp(1);                    // 2.7182pow(E,x)
Math.random();                  // random number between 0 and 1
Math.floor(Math.random() * 5) + 1;  // random integer, from 1 to 5
```

Array:

```
var flowers = ["Rose", "Sunflower", "Lotus", “Lily”];                    // Array Declaration
var flowers = new Array ("Rose", "Sunflower", "Lotus", “Lily”);   //Alternate method.
alert(flowers[1]);                                                  // access value at index.
dogs[0] = "Hibiscus";                                               // change the first item to “Hibiscus”.
for (var i = 0; i < flowers.length; i++) {                          // Accessing array element using loop.
        console.log(flowers[i]);
}
```

Array Methods:

```
flowers.toString();                                  // convert the array to string.
flowers.join(" _ ");                                 // join between two array element.
flowers.pop();                                       // remove last element.
flowers.push("Tulip");                               // add new element to the end.
flowers [flowers.length] = "Tulip";                  // the same as push.
flowers.shift();                                     // remove first element.
flowers.unshift("Tulip");                            // add new element to the beginning.
delete.flowers[0];                                   // change element to undefined.
flowers.splice(2, 0, "ABC", "DEF");                  // add elements.
var f = flowers.concat(a,b);                         // join two arrays (a followed by b).
flowers.slice(1,4);                                  // elements from [1] to [4-1].
flowers.sort();                                      // sort string alphabetically.
flowers.reverse();                                   // sort string in descending order.
x.sort(function(a, b){return a - b});                        // numeric sort.
x.sort(function(a, b){return b - a});                        // numeric descending sort.
x.sort(function(a, b){return 0.5 - Math.random()});          // random order sorting.
```

Regular Expressions:

```
var a = str.search(/CheatSheet/i);
```

Modifiers:

- **i** perform case-insensitive matching.
- **g** perform a global match.
- **m** perform multiline matching.

Patterns:

- **** Escape character
- **\d** find a digit
- **\s** find a whitespace character
- **\b** find match at beginning or end of a word
- **n+** contains at least one n
- **n*** contains zero or more occurrences of n
- **n?** contains zero or one occurrences of n
- **^** Start of string
- **$** End of string
- **\uxxxx** find the Unicode character
- **.** Any single character
- **(x | y)** x or y
- **(...)** Group section
- **[xyz]** In range (x, y or z)
- **[0-9]** any of the digits between the brackets
- **[^xyz]** Not in range
- **\s** White space

- **a?** Zero or one of a
- **a*** Zero or more of a
- **a*?** Zero or more, ungreedy
- **a+** One or more of a
- **a+?** One or more, ungreedy
- **a{2}** Exactly 2 of a
- **a{2,}** 2 or more of a
- **a{,10}** Up to 10 of a
- **a{1,6}** 1 to 6 of a
- **a{4,6}?** 4 to 6 of a
- **[:punct:]** Any punctuation symbol
- **[:space:]** Any space character
- **[:blank:]** Space or tab

If–Else Statements:

```
if ((age >= 10) && (age < 20)) {          // logical condition
    status = "Permitted.";                // executed if condition is true
}
else {                                    // else block is optional
    status = "Not Permitted.";            // executed if condition is false
}
```

Switch Statement:

```
switch (day) {                          // Input is current day in numeric.
case 1:                                 // if (day == 1)
        text = "Monday";
        break;
case 2:                                 // if (day == 2)
        text = "Tuesday";
        break;
case 3:                                 // if (day == 3)
        text = "Wednesday";
        break;
case 4:                                 // if (day == 4)
        text = "Thursday";
        break;
case 5:                                 // if (day == 5)
        text = "Friday";
        break;
case 6:                                 // if (day == 6)
        text = "Saturday";
        break;
case 7:                                 // if (day == 7)
        text = "Sunday";
        break;
default:                                // else....
        text = "Please enter valid day number.";
   }
```

For Loop:

```
for (var i = 0; i < 5; i++) {
        document.write(i + ": " + i*i+ "<br/>");
}
```

```
var sum = 0;
for (var i = 0; i < a.length; i++) {
        sum + = a[i];
}
```

```
html = "";
for (var i of stud) {
        html += "<li>" + i + "</li>";
}
```

While Loop:

```
var i = 1;                          // initialize
while (i < 20) {                    // enters the cycle if statement is true
        i += i;                     // increment to avoid infinite loop
        document.write(i + ", ");   // output
}
```

Do While Loop:

```
var i = 1;                          // initialize
do {                                // enters loop at least once
      i += 1;                       // increment to avoid infinite loop
document.write(i + ", ");           // output
} while (i < 20)                    // repeats loop, if statement is true
```

Break Statement:

```
for (var i = 0; i < 20; i++) {
        if (i == 10) {
                break;                          // terminates the loop
        }
        document.write(i + ", ");
}
```

Continue Statement:

```
for (var i = 0; i < 20; i++) {
        if (i == 10) {
                continue;                               // skips
        }
        document.write(i + ", ");
}
```

Dates:

```
var d = new Date();
a = d.getDay();                          // getting the weekday
getDate();                               // day as a number (1-31)
getDay();                                // weekday as a number (0-6)
getFullYear();                           // four digit year (yyyy)
getHours();                              // hour (0-23)
getMilliseconds();                       // milliseconds (0-999)
getMinutes();                            // minutes (0-59)
getMonth();                              // month (0-11)
getSeconds();                            // seconds (0-59)
getTime();                               // milliseconds since 1970
d.setDate(d.getDate() + 7);              // adds a week to a date
setDate();                               // day as a number (1-31)
setFullYear();                           // year (optionally month and day)
setHours();                              // hour (0-23)
setMilliseconds();                       // milliseconds (0-999)
setMinutes();                            // minutes (0-59)
setMonth();                              // month (0-11)
setSeconds();                            // seconds (0-59)
setTime();                               // milliseconds since 1970)
```

Global Functions:

```
eval();                          // executes a string as if it was scriptcode
String(44);                      // return string from number
(44).toString();                 // return string from number
Number("44");                    // return number from string
decodeURI(enc);                  // decode URI. Result: "page.asp"
encodeURI(uri);                  // encode URI. Result: "page.asp"
decodeURIComponent(enc);         // decode a URI component
encodeURIComponent(uri);         // encode a URI component
parseFloat();                    // returns floating point number of string
parseInt();                      // parses a string and returns an integer
isFinite();                      // is variable a finite, legal number
isNaN();                         // is variable an illegal number
```

Events:

```
<button onclick="myFunction();"> Click here </button>
```

- **Mouse Events:**
  - onclick
  - oncontextmenu
  - ondblclick
  - onmousedown
  - onmouseenter
  - onmouseleave
  - onmousemove
  - onmouseover
  - onmouseout
  - onmouseup

- **Keyboard Events:**
  - onkeydown
  - onkeypress
  - onkeyup
- **Frame Events:**
  - onabort
  - onbeforeunload
  - onerror
  - onhashchange
  - onload
  - onpageshow
  - onpagehide
  - onresize,
  - onscroll
  - onunload
- **Form Events:**
  - onblur
  - onchange
  - onfocus
  - onfocusin
  - onfocusout
  - oninput
  - oninvalid
  - onreset
  - onsearch
  - onselect
  - onsubmit

- **Drag Events:**
  - ondrag
  - ondragend
  - ondragenter
  - ondragleave
  - ondragover
  - ondragstart
  - ondrop
- **Clipboard Events:**
  - oncopy
  - oncut
  - onpaste
- **Media Events:**
  - onabort
  - oncanplay
  - oncanplaythrough
  - ondurationchange
  - onended
  - onerror
  - onloadeddata
  - onloadedmetadata
  - onloadstart
  - onpause
  - onplay
  - onplaying
  - onprogress
  - onratechange
  - onseeked
  - onseeking
  - onstalled
  - onsuspend

- ontimeupdate
- onvolumechange
- onwaiting

- **Animation Events:**
  - Animationend
  - Animationiteration
  - Animationstart

- **Miscellaneous Events:**
  - transitioned
  - onmessage
  - onmousewheel
  - ononline
  - onoffline
  - onpopstate
  - onshow
  - onstorage
  - ontoggle
  - onwheel
  - ontouchcancel
  - ontouchmove

Errors:

```
try {                                   // try block of code
        undefinedFunction();
}
catch(err) {                            // block to handle errors
        console.log(err.message);
}
```

Throw error:

```
throw "My error message";                    // throws a text
```

Input validation:

```
const input = document.getElementById("num");      // get input element
try {
    const x = input.value;                         // get input value
    if (x == "") throw "empty";                    // error cases
    if (isNaN(x)) throw "not a number";
    x = Number(x);
    if (x > 20) throw "too high";
} catch (err) {                                    // if there's an error
    console.log(`Input is ${err}`);                // output error
    console.error(err);                            // write the error in console
} finally {
    console.log("Done");              // executed regardless of the try catch block
}
```

Error Names:

- **RangeError:** A number is "out of range".
- **ReferenceError:** An illegal reference has occurred.
- **SyntaxError:** A syntax error has occurred.
- **TypeError:** A type error has occurred.
- **URIError:** An encodeURI() error has occurred.

JSON: JavaScript Object Notation

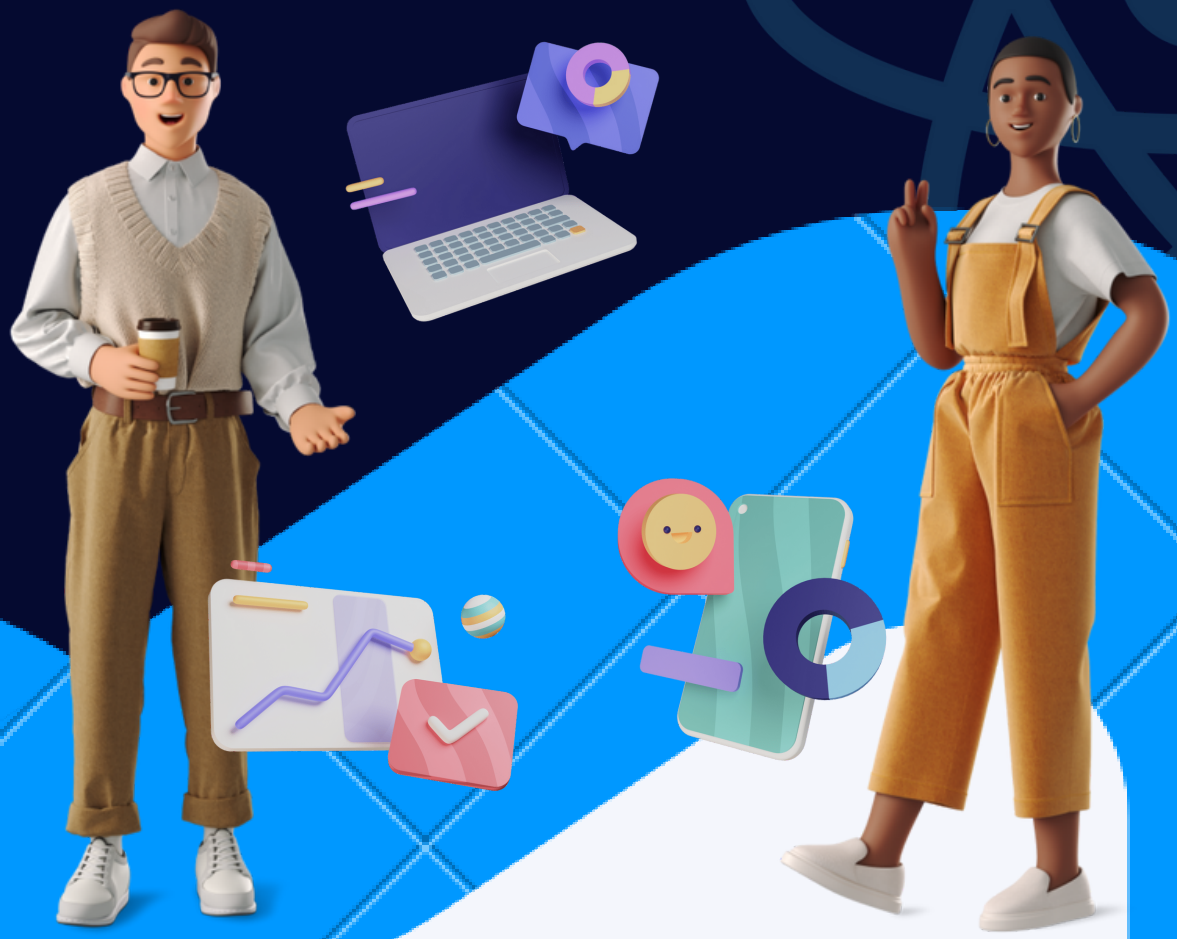
```
let str = '{"names":[' +                                    // create JSON object
'{"first":"Koushik","lastN":"Sadhu" },' +
'{"first":"Pranay","lastN":"Gupta" },' +
'{"first":"Shuvam","last":"Chodhury" }]}';
obj = JSON.parse(str);
console.log(obj.names[1].first);

// Send:

let myObj = { "name":"Nidhi", "age":20, "city":"Kolkata" };      // create object
let myJSON = JSON.stringify(myObj);                              // stringify
window.location = `demo.html?x=${myJSON}`;                       // sending to php

// Storing and retrieving:

let myObj = { "name":"Nidhi", "age":20, "city":"Kolkata" };
let myJSON = JSON.stringify(myObj);                              // storing data
localStorage.setItem("testJSON", myJSON);
let text = localStorage.getItem("testJSON");                     // retrieving data
let obj = JSON.parse(text);
document.write(obj.name);
```

Promises:

```
function sum(a, b) {
   return new Promise((resolve, reject) => {
      setTimeout(() => {                                  // send the response after 1 second
         if (typeof a !== "number" || typeof b !== "number") {     // testing input types
         return reject(new TypeError("Inputs must be numbers"));
        }
   resolve(a + b);
   }, 1000);
 });
}
let myPromise = sum(10, 5);
myPromise.then((result) => {
console.log("10 + 5: ", result);
   return sum(null, "foo");                    // Invalid data and return another promise
 })
 .then(() => {                                 // Won't be called because of the error
 })
 .catch((err) => {
   console.error(err); // => Inputs must be numbers
});
```

React

CheatSheet



CODE HELP

In this Cheatsheet, we will cover the basics of React. We will provide examples to help you understand how React works and how to use them in your own web development projects. Whether you are a beginner or an experienced developer, this PDF can serve as a useful reference guide.

React is a JavaScript library for building user interfaces. It was developed and is maintained by Facebook and a community of individual developers and companies. React allows developers to build reusable UI components, manage the state of their applications, and render dynamic updates efficiently.

React uses a virtual DOM (Document Object Model) to update the view without having to reload the entire page. This results in fast and smooth updates to the user interface. React also uses a component-based architecture, which allows developers to easily reuse and compose their components to build complex UIs.

React is used for both web and mobile development, and is popular for its simplicity, versatility, and performance. Many companies and organizations use React for their websites and applications, including Facebook, Airbnb, Netflix, and Dropbox.

MANAGE STATE

useState

```
const [count, setCount] = useState(initialCount);
```

- Class way

```
const initialCount = 0;
class Counter extends Component {
      constructor(props) {
            super(props);
            this.state = { count : initialCount };
      }
      render() {
         return (
         <div>
            <p>You clicked {this.state.count} times</p>
            <button
            onClick = {() => this.setState(({count}) => ({ count: count + 1 }))}
            >
            Click me
            </button>
         </div>
      );
   }
}
```

- Hook Way

```
import { useState } from "react";
const initialCount = 0;
function Counter() {
      const [count, setCount] = useState(initialCount);
      return (
      <div>
            <p>You clicked {count} times</p>
            <button
            onClick = {()  =>  setCount((c) => c + 1)}
            >
                  Click me
            </button>
      </div>
   );
}
```

useReducer

```
const [state, dispatch] = useReducer(
      reducer,
      initialState,
      initialDispatch
);
```

An alternative to useState. Use it in the components that need complex state management, such as multiple state values being updated by multiple methods.

```
const initialState = {count: 0};
function reducer(state, action) {
      switch (action.type) {
            case 'increment':
                  return {count: state.count + 1};
            case 'decrement':
                  return {count: state.count - 1};
            default:
                  throw new Error();
            }
      }
function Counter({initialState}) {
      const [state, dispatch] = useReducer(reducer, initialState);
      return (
      <>
      Count: {state.count}
      <button onClick = {() => dispatch({type: 'increment'})}>+</button>
      <button onClick = {() => dispatch({type: 'decrement'})}>+</button>
      </>
    );
}
```

HANDLE SIDE EFFECTS

useEffect

```
useEffect(() => {
      applyEffect(dependencies);
      return () => cleanupEffect();
}, [dependencies]);
```

- Class way

```
class FriendStatus extends React.Component {
	state = { isOnline : null };
	componentDidMount() {
		ChatAPI.subscribeToFriendStatus(
		this.props.friend.id,
		this.handleStatusChange
		);
	}
	componentWillUnmount() {
		ChatAPI.unsubscribeFromFriendStatus(
		this.props.friend.id,
		this.handleStatusChange
		);
	}
	componentDidUpdate(prevProps) {
		if(this.props.friend.id !== prevProps.id) {
			ChatAPI.unsubscribeFromFriendStatus(
			prevProps.friend.id,
			this.handleStatusChange
			);
			ChatAPI.subscribeToFriendStatus(
			this.props.friend.id,
			this.handleStatusChange
			);
		}
	}
	handleStatusChange = status => {
		this.setState({
		isOnline: status.isOnline
	});
 }
render() {
	if (this.state.isOnline === null) {
		return 'Loading...';
	}
	return this.state.isOnline ? 'Online' : 'Offline';
	}
}
```

- Hook Way

```
import { useState, useEffect } from 'react';
function FriendStatus(props) {
      const [isOnline, setIsOnline] = useState(null);
      status
      const handleStatusChange = (status) => setIsOnline(status.isOnline);
      useEffect(
      () => {
 // Update status when the listener triggers
      ChatAPI.subscribeToFriendStatus(
      props.friend.id,
      handleStatusChange
      );
 // Stop listening to status changes every time we cleanup
       return function cleanup() {
            ChatAPI.unsubscribeFromFriendStatus(
            props.friend.id,
            handleStatusChange
            );
      };
 },
 [props.friend.id] // Cleanup when friend id changes or when we "unmount"
 );
if (isOnline === null) {
      return 'Loading...';
 }
return isOnline ? 'Online' : 'Offline';
}
```

useLayoutEffect

```
useLayoutEffect(() => {
      applyBlockingEffect(dependencies);
      return cleanupEffect();
}, [dependencies]);
```

useLayoutEffect is almost same as useEffect, but fires synchronously after the render phase. Use this to safely read from or write to the DOM

```
import { useRef, useLayoutEffect } from "react";
function ColoredComponent({color}) {
	const ref = useRef();
	useLayoutEffect(() => {
		const refColor = ref.current.style.color;
		console.log(`${refColor} will always be the same as ${color}`);
		ref.current.style.color = "rgba(255,0,0)";
	}, [color]);
	useEffect(() => {
		const refColor = ref.current.style.color;
		console.log(
		`but ${refColor} can be different from ${color} if you play with the
DOM`
		);
	}, [color]);
	return (
		<div ref={ref} style={{ color: color }}>
			Hello React hooks !
		</div>
	);
}
```

```
<ColoredComponent color = {"rgb(42, 13, 37)"}/>
// rgb(42, 13, 37) will always be the same as rgb(42, 13, 37)
// but rgb(255, 0, 0) can be different from rgb(42, 13, 37) if you play with the DOM
```

USE THE CONTEXT API

useContext

```
const ThemeContext = React.createContext();
const contextValue = useContext(ThemeContext);
```

- Class way

```
class Header extends React.Component {
      public render() {
      return (
      <AuthContext.Consumer>
      {({ handleLogin, isLoggedIn}) => (
      <ModalContext.Consumer>
      {({ isOpen, showModal, hideModal }) => (
      <NotificationContext.Consumer>
      {({ notification, notify }) => {
            return (
            ...
            )
      }}
      </NotificationContext.Consumer>
      )}
      </ModalContext.Consumer>
      )}
      </AuthContext.Consumer>
      );
   }
}
```

- Hook Way

```
import { useContext } from 'react';
function Header() {
      const { handleLogin, isLoggedIn} = useContext(AuthContext);
      const { isOpen, showModal, hideModal } = useContext(ModalContext);
      const { notification, notify } = useContext(NotificationContext);

      return ...
}
```

NOTE: Use the created context, not the consumer.

```
const Context = React.createContext(defaultValue);
// Wrong
const value = useContext(Context.Consumer);
// Right
const value = useContext(Context);
```

MEMOIZE EVERYTHING

useMemo

```
const memoizedValue = useMemo (
      () => expensiveFn (dependencies),
      [dependencies]
);
```

```
function RandomColoredLetter(props) {
        const [color, setColor] = useState('#fff')
        const [letter, setLetter] = useState('a')
        const handleColorChange = useMemo(() => () =>
setColor(randomColor()), []);
        const handleLetterChange = useMemo(() => () =>
setLetter(randomColor()), []);
        return (
    <div>
        <ColorPicker handleChange={handleColorChange} color={color} />
        <LetterPicker handleChange={handleLetterChange} letter={letter} />
        <hr/>
        <h1 style={{color}}>{letter}</h1>
    </div>
  )
 }
```

useCallback

```
const  memoizedCallback  =  useCallback (
      expensiveFn (dependencies),
      [dependencies]
);
```

```
function RandomColoredLetter(props) {
   const [color, setColor] = useState('#fff')
   const [letter, setLetter] = useState('a')
   const handleColorChange = useCallback(() => setColor(randomColor()), []);
   const handleLetterChange = useCallback(() => setLetter(randomColor()), []);
   return (
   <div>
        <ColorPicker handleChange={handleColorChange} color={color} />
        <LetterPicker handleChange={handleLetterChange} letter={letter} />
        <hr/>
        <h1 style={{color}}>{letter}</h1>
   </div>
  )
 }
```

USE REFS

useRef

```
const ref = useRef();
```

useRef can just be used as a common React ref :

```
import { useRef } from "react";
function TextInput() {
      const inputRef = useRef(null);
      const onBtnClick = () => inputRef.current.focus();
      return (
      <>
      <input ref={ inputRef } />
      <button onClick={onBtnClick}>Focus the text input</button>
      </>
   )
}
```

But it also allows you to just hold a mutable value through any render. Also, mutating the value of ref.current will not cause any render.

useImperativeHandle

```
useImperativeHandle (
      ref,
      createHandle,
      [dependencies]
)
```

useImperativeHandle allows you to customize the exposed interface of a component when using a ref. The following component will automatically focus the child input when mounted :

```
function TextInput(props, ref) {
	const inputRef = useRef(null);
	const onBtnClick = () => inputRef.current.focus();
	useImperativeHandle(ref, () => ({
		focusInput: () => inputRef.current.focus();
	});
	return (
	<Fragment>
	<input ref={inputRef} />
	<button onClick={onBtnClick}>Focus the text input</button>
	</Fragment>
	)
}
const TextInputWithRef = React.forwardRef(TextInput);
function Parent() {
	const ref = useRef(null);
	useEffect(() => {
		ref.focusInput();
	}, []);
	return (
	<div>
	<TextInputWithRef ref={ref} />
	</div>
	);
}
```

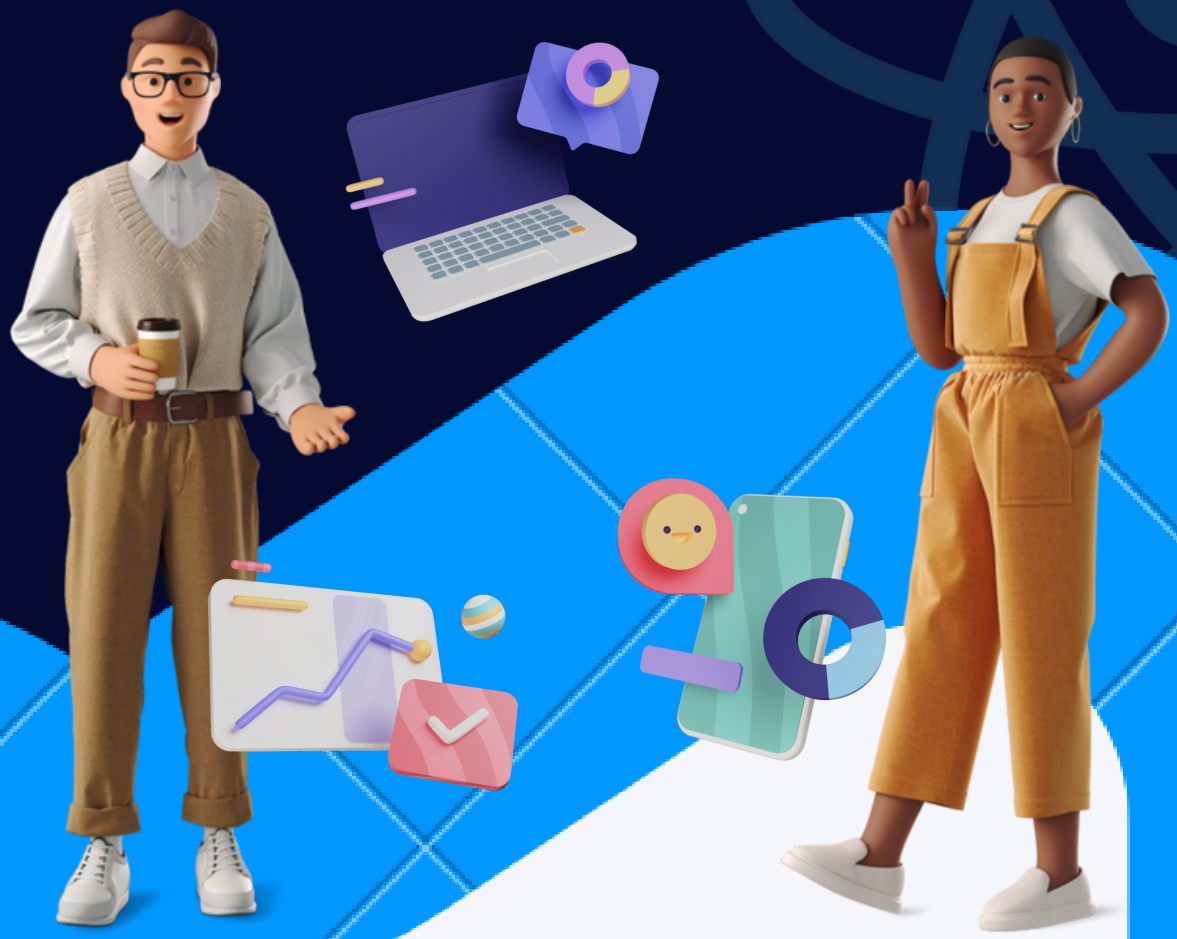
REUSABILITY

Extract reusable behaviour into custom hooks.

```
import { useState, useRef, useCallback, useEffect } from "React";
// let's hide the complexity of listening to hover changes
function useHover() {
	const [value, setValue] = useState(false); // store the hovered state
	const ref = useRef(null); // expose a ref to listen to
 // memoize function calls
	const handleMouseOver = useCallback(() => setValue(true), []);
	const handleMouseOut = useCallback(() => setValue(false), []);
 // add listeners inside an effect,
 // and listen for ref changes to apply the effect again
	useEffect(() => {
		const node = ref.current;
		if (node) {
			node.addEventListener("mouseover", handleMouseOver);
			node.addEventListener("mouseout", handleMouseOut);
		return () => {
			node.removeEventListener("mouseover", handleMouseOver);
			node.removeEventListener("mouseout", handleMouseOut);
			};
		}
	}, [ref.current]);
 // return the pair of the exposed ref and it's hovered state
	return [ref, value];
}
```

```
const HoverableComponent = () => {
	const [ref, isHovered] = useHover();
	return (
	<span style={{ color: isHovered ? "blue" : "red" }} ref={ref}>
		Hello React hooks !
	</span>
	);
};
```

Express CheatSheet



CODE HELP

In this Cheatsheet, we will cover the basics of Express.js. We will provide examples to help you understand how Express.js works and how to use them in your own web development projects. Whether you are a beginner or an experienced developer, this PDF can serve as a useful reference guide.

Express.js is a popular open-source web application framework for Node.js, a JavaScript runtime that allows developers to build scalable, high-performance server-side applications. It provides a wide range of features and utilities for web application development, such as routing, middleware, and template rendering.

With Express.js, developers can quickly create server-side applications that handle HTTP requests, process data, and respond with dynamic content or data from a database. It also supports a wide range of middleware that can be used to add functionality, such as authentication, compression, logging, and more.

Express.js is designed to be flexible and easy to use, allowing developers to customize and extend it to meet the specific needs of their applications. Its modular structure and extensive community support make it one of the most popular frameworks for building web applications with Node.js.

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.

INSTALLATION

```
npm install express
```

HELLO WORLD SERVER

```
const express = require('express');
const app = express();
app.set('port', (process.env.PORT || config.port));
app.get('/', (req, res) => res.send('Hello World!'));
app.listen(app.get('port'), () =>
    console.log(`Server started on ${app.get('port')} port`))
```

BASIC ROUTING:

```
// GET
app.get('/', function (req, res) {
    res.send('Hello World!')
})

// POST
app.post('/', function (req, res) {
    res.send('POST request. body:', req.body)
})

// DELETE
app.delete('/:id', function (req, res) {
    res.send('DELETE request for id:'. req.params.id)
})
```

STATIC FILE SERVING:

```
app.use(express.static(__dirname + '/public'));
```

LOGGING ALL ROUTES:

```
app._router.stack.forEach(function(r) {
    if (r.route && r.route.path) {
        console.log(r.route.path)
    }
});
```

DEFINING ROUTES IN A DIFFERENT FILE:

File: /routes/users.js

```
// File Path: /routes/users.js

var express = require('express');
var router = express.Router();
router.get('/', (req, res) => {
    const users = [];                    // get from db
    res.send(users);
});
router.get('/:id', (req, res) => {
    const user = {};                     // get from db
    res.send(user);
});
router.post('/', (req, res) => {
    const user = req.body;               // save user to db
    res.send({status: 'success'});
});
module.exports = router;
```

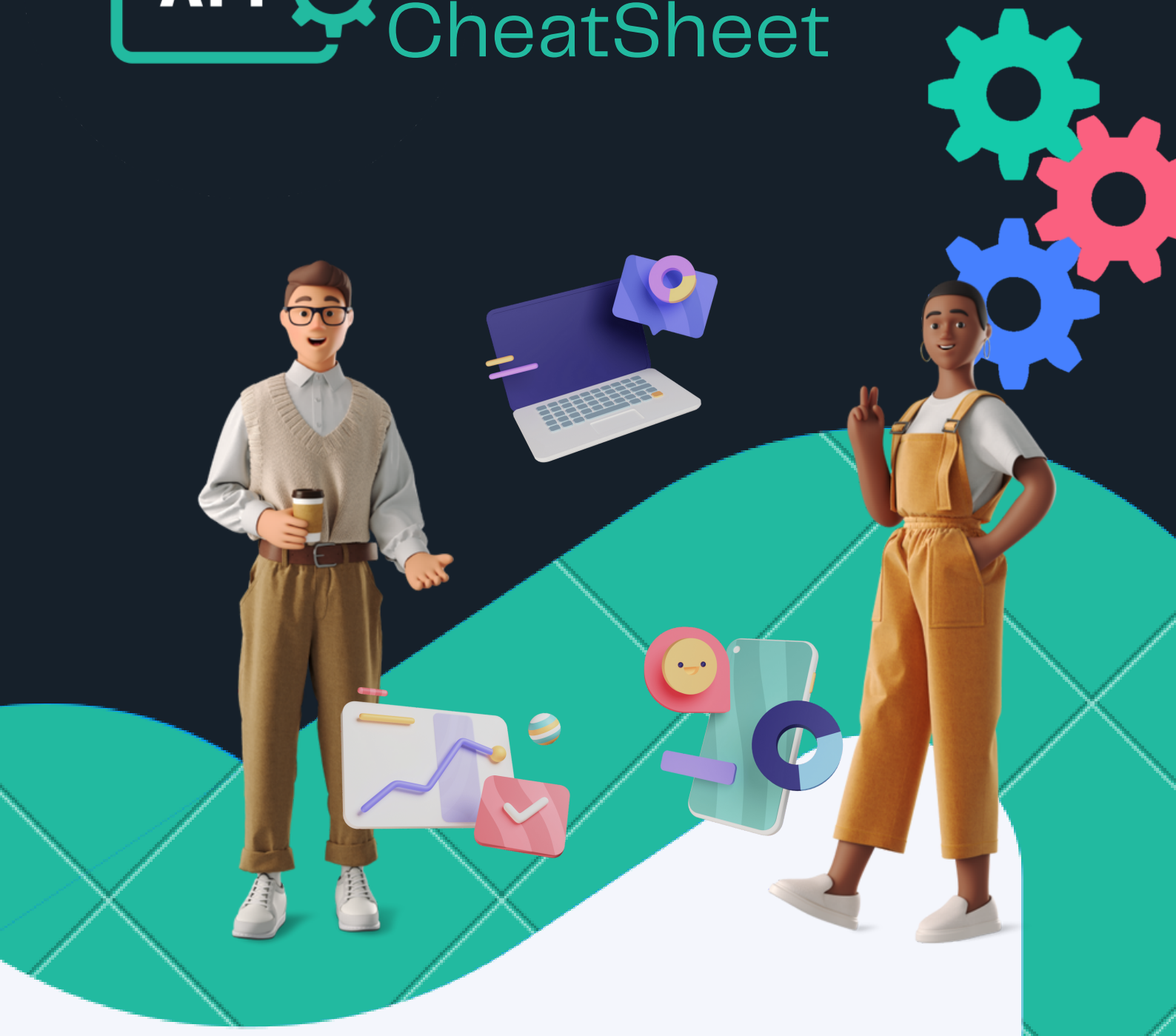
ADDING ROUTES FROM: /routes/users.js

```
app.use('/user', require('./routes/user'));
```

REDIRECTS:

```
router.get('/old-path', function(req, res) {
      res.redirect('/temp-new-path');                    // sends a 302
});

router.get('/old-path', function(req, res) {
      res.redirect(301, '/permanent-new-path');          // sends a 301
});
```

APIs CheatSheet



CODE HELP

In this Cheatsheet, we will cover the basics of API. We will provide examples to help you understand how API's work and how to use them in your own web development projects. Whether you are a beginner or an experienced developer, this PDF can serve as a useful reference guide.

An API, or Application Programming Interface, is a set of rules and protocols for building and interacting with software applications. It allows different software systems to communicate with each other, enabling them to share data and functionality. This allows developers to access the functionality of a certain software application or system without having to understand its underlying code or implementation.

An example of an API is the Facebook API, which allows developers to access and interact with the functionality of the Facebook platform, such as posting status updates, retrieving user information, and managing ad campaigns. Another example is the Google Maps API, which allows developers to embed maps and location-based functionality in their own websites and apps.

How an API Works:

APIs act as a bridge between applications and web servers, processing data transfer between systems. When a client application initiates an API call, also known as a request, it is sent to the web server via the API's Uniform Resource Identifier (URI) and includes a request verb, headers, and sometimes a request body. The API then processes the request and may make a call to an external program or web server for the requested information.

The server responds with the requested data, which the API then forwards to the initial requesting application. This process of requests and responses all happens through the API. Unlike user interfaces which are designed for human use, APIs are designed for use by computers or applications.

REST API: (Representational State Transfer)

REST (Representational State Transfer) is a type of web architecture and a set of constraints to be used when creating web services. RESTful API (Application Programming Interface) is an API that conforms to the REST architectural style and constraints, and it is typically used to make requests to retrieve or update data on a web server. A RESTful API uses HTTP requests to POST (create), PUT (update), GET (read), and DELETE (delete) data. A RESTful API also returns a response in a standard format, typically JSON or XML, and uses standard HTTP status codes to indicate the status of the request. RESTful APIs are popular because they are simple to understand and easy to use, and they work well with the HTTP protocol that the internet is built on. Additionally, RESTful APIs are often faster and more lightweight than their SOAP (Simple Object Access Protocol) counterparts because they use smaller message formats. RESTful API's have become a popular way for systems to expose databases through HTTP(S) following CRUD operations (Create, Read, Update, Delete), and return JSON or XML as responses, it's also widely used in microservices, mobile and web applications, IoT, and many more.

REST requires that a client make a request to the server in order to retrieve or modify data on the server.

A request generally consists:

- An HTTP verb, which defines what kind of operation to perform.
- A header, which allows the client to pass along information about the request.
- A path to a resource.
- An optional message body containing data.

CRUD: (Create Read Update Delete)

CRUD stands for Create, Read, Update, and Delete, which are the four basic operations that can be performed on data in a database. These operations are often used to interact with databases through APIs (Application Programming Interfaces).

- **Create:** This operation is used to add new data to the database. It is typically performed using the **HTTP POST** method and is used to create new resources.

- **Read:** This operation is used to retrieve data from the database. It is typically performed using the **HTTP GET** method and is used to read existing resources.

- **Update:** This operation is used to modify existing data in the database. It is typically performed using the **HTTP PUT** method and is used to update existing resources.

- **Delete:** This operation is used to remove data from the database. It is typically performed using the **HTTP DELETE** method and is used to delete resources.

CRUD operations are typically implemented in RESTful APIs, but they can also be used in other types of APIs. These operations can be used to expose the data in a database to other systems, such as a mobile app or a web application, that can use the data to provide a service to end-users.

HTTP Verbs

In the context of API (Application Programming Interface), HTTP verbs (or methods) are used to specify the type of action that the API client wants to perform on a resource. The most commonly used HTTP verbs in API are:

- **GET:** This verb is used to retrieve information from the server. It is the most common verb used to make a request to an API. It is considered safe, meaning that it should not have any side effects on the server or the data it serves.

- **POST:** This verb is used to submit information to the server. It is typically used to create new resources.

- **DELETE:** This verb is used to delete resources.

- **PUT:** This verb is used to update existing resources. It replaces the entire resource, unlike the PATCH verb which modifies only the fields sent in the request.

- **PATCH:** This verb is used to partially update a resource. It is used to modify only the fields sent in the request and it's usually used when you don't want to replace all the resource's attributes.

- **PUT vs PATCH:** PUT method uses the request URI to supply a modified version of the requested resource which replaces the original version of the resource, whereas the PATCH method supplies a set of instructions to modify the resource.

- **HEAD:** This verb is similar to GET, but it only returns the headers of the response, without the body.

- **OPTIONS:** This verb is used to retrieve the allowed actions on a resource.

- **CONNECT:** Connect request establishes a tunnel to the server identified by a specific URI. A good example is SSL tunnelling.

- **TRACE:** The Trace method performs a message loop-back test along the path to the target resource, to provide a useful debugging mechanism. It allows clients to view whatever message is being received at the other end of the request chain so that they can use the info for testing or diagnostic functions.

These verbs are part of the HTTP protocol, and are commonly used in RESTful (Representational State Transfer) APIs, which are built on top of the HTTP protocol and are designed to be easily consumed by web and mobile applications.

HTTP Status Codes

HTTP status codes are three-digit numbers returned by an API (Application Programming Interface) to indicate the status of a request. These codes provide information about whether a request was successful or not, and they are an important part of the API development.

The first digit of the status code specifies one of five standard classes of responses.

Standard Classes:

- **1xx:** Informational responses.
- **2xx:** Successful responses.
- **3xx:** Redirection responses.
- **4xx:** Client error responses.
- **5xx:** Server error responses.

1xx: Informational responses

It indicates that the request was received and understood by the server and it's continuing the process.

- **100:** Continue.
- **101:** Switching Protocols.
- **102:** Processing.
- **103:** Early Hints.

2xx: Successful responses

It indicates that the action requested by the client was received, understood, and accepted.

- **200:** OK.
- **201:** Created.
- **202:** Accepted.
- **203:** Non-Authoritative Information.
- **204:** No Content.

3xx: Redirection responses

Many of these 3xx status codes are used in URL redirection or it indicates the client must take additional action to complete the request.

- **301:** Moved Permanently.
- **302:** Found.
- **304:** Not Modified.
- **305:** Use Proxy.
- **307:** Temporary Redirect.
- **308:** Permanent Redirect.

4xx: Client error responses

This status code is intended for situations in which the error seems to have been caused by the client.

- **400:** Bad Request.
- **401:** Unauthorized.
- **403:** Forbidden.
- **404:** Not Found.
- **406:** Not Acceptable.
- **408:** Request Timeout.

5xx: Server error responses

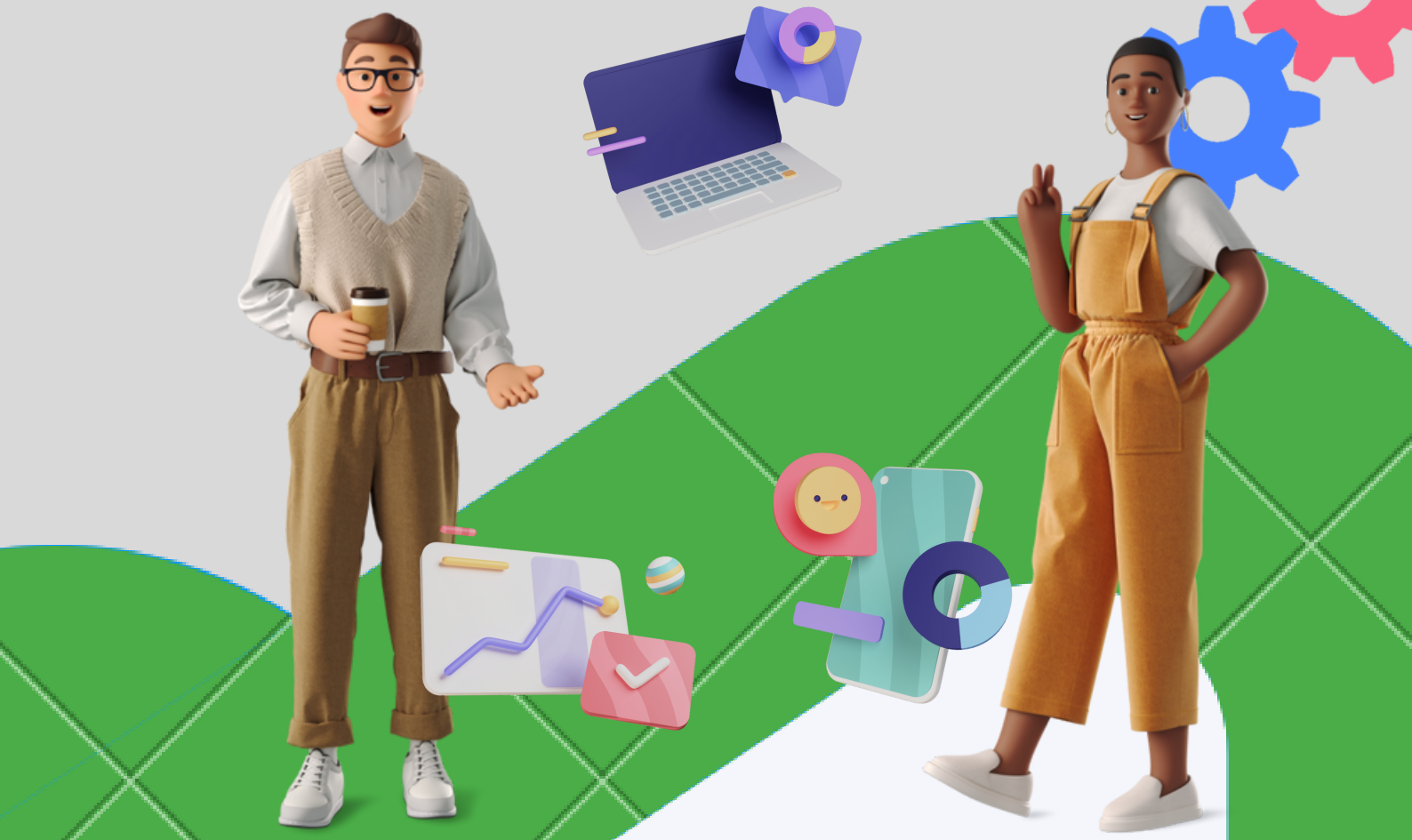
It indicates that the server has encountered a situation where it doesn't know how to handle a request.

- **500:** Internal Server Error
- **501:** Not Implemented
- **502:** Bad Gateway
- **503:** Service Unavailable
- **504:** Gateway Timeout
- **505:** HTTP Version Not Supported

These are just a few examples of the many HTTP status codes that can be returned by an API. It's important to understand the meaning of each code and to handle them appropriately in the client application. The HTTP status codes are standardized and are used as a way for the server to communicate with the client about the outcome of a request.

mongo DB

CheatSheet



CODE HELP

In this Cheatsheet, we will cover the basics of mongo DB. We will provide examples to help you understand how mongo DB work and how to use them in your own web development projects. Whether you are a beginner or an experienced developer, this PDF can serve as a useful reference guide.

MongoDB is a document-oriented, open-source database program that is used for storing and retrieving data. It is a NoSQL database, which means that it does not use a fixed schema and does not rely on tables to organize data, unlike a traditional relational database. Instead, MongoDB stores data in a format called BSON (binary JSON), which allows for more flexibility and scalability when working with large amounts of data. MongoDB is often used for web and mobile applications, real-time analytics, and big data processing.

Most Commonly Used MongoDB terms:

Common DB Terms	MongoDB Terms
Database	Database
Tables	Collections
Rows	Document
Columns	Fields

NOTE: In MongoDB, data are store in **BSON** not in **JSON** format.

ATOMIC/UPDATE OPERATOR

Atomic Operator	Use of the Operator
$set	Updating the stored document without changing the prev. state of the document.
$inc	To increment and decrement on the numerical type variable.
$push	Push new field but it must be of ARRAY type field.
$eq	Matches values that are equal to the given value.
$gt	Matches if values are greater than the given value.
$lt	Matches if values are less than the given value.
$gte	Matches if values are greater or equal to the given value.
$lte	Matches if values are less or equal to the given value.
$in	Matches any of the values in an array.
$ne	Matches values that are not equal to the given value.
$nin	Matches none of the values specified in an array.

MongoDB COMMANDS

- For invoking Mongo Daemon:

```
mongod
```

- For starting the mongo shell:

```
mongo
```

- To show all the database present in the MongoDB:

```
show dbs
```

- To create a new database:

```
use students
```

- To see the current database:

```
db
```

- To delete the current database which we are in:

```
db.dropDatbase()
```

- To show all the collection in the database:

```
show collections
```

- To show all the collection in the database:

```
show collections
```

- To create a new collections:

```
db.createCollection('studentData')
```

- To delete a collection:

```
db.studentData.drop()
```

- To insert a document in the collections:

```
db.studentData.insert({name : "Jack", age : 20})
```

- For inserting only one document into the collections (same work as " .insert() "):

```
db.studentData.insertOne({
        name : "Travour",
        email : "travor458@gmail.com",
        age : 24
})
```

- For inserting multiple document into the collections:

```
db.studentData.insertMany([
   {name : "Harry", email : "marry234@gmail.com", age : 13},
   {name : "Ronit", email : "ronit@outlook.com", age : 45},
   {name : "Adesh", email : "sadhuadesh@gmail.com", age : 18}
])
```

- For displaying the documents present inside the collections:

```
db.studentData.find()
```

- For displaying the documents present inside the collections in a prettyer format:

```
db.studentData.find().pretty()
```

- For displaying the limited number of document present in the collection:

```
db.studentData.find().limit(3).pretty()
```

- For counting the number of document present in the collections:

```
db.studentData.find().count()
```

- For sorting the document in ascending order on the basis of age:

```
db.studentData.find().sort({age : 1})
```

- For sorting the document in descending order on the basis of age:

```
db.studentData.find().sort({age : -1})
```

- For chaining multiple function together:

```
db.studentData.find().limit(2).sort({age : -1}).pretty()
```

- For finding only one document:

```
db.studentData.findOne({age : 18})
```

- For finding a group of documents based on some condition:

```
db.studentData.findOne({age : {$lte : 18}})
```

- For updating a data based upon certain condition or filter: (This will change the entire documents and fields by removing the previous state of the document)

```
db.studentData.update({name : "Harry"} , {fblogged : "Yes" , age : 45})
```

- For updating a data based upon certain condition or filter: (This will change the documents and fields without changing the previous state of the document)

```
db.studentData.updateOne({name : "Harry"} , {$set : {fblogged : "Yes" , age : 45}})
```

- For updating a group of document based on certain condition:

```
db.studentData.updateMany({age : 24} , {$set : {courseCount : 2}})
```

- For adding a new field to all the documents:

```
db.studentData.updateMany({} , {$set : {registered : "Yes"}})
```

- For renaming a field name:

```
db.studentData.updateMany({} , {$rename : {age : "student_age"}})
```

- For incrementing a numeric field in a document:

```
db.studentData.updateMany({} , {$inc : {age : 1}})
```

- For decrementing a numeric field in a document:

```
db.studentData.updateMany(
        {age : {$lt : 19}} ,
        {
                $inc : {courseCount : -1},
                $set {hobby : ["Painting", "Football"]}
        })
```

- To append new data to the array type field:

```
db.studentData.updateMany(
    {age : {$lt : 19}} ,
    {
        $push : {hobby : "Swimming"}
    })
```

- To delete One document using condition or filter:

```
db.studentData.deleteOne({age : 17})
```

- To delete the first document from the table:

```
db.studentData.deleteOne({})
```

- To delete the first document from the table:

```
db.studentData.deleteMany({fblogged : "Yes"})
```

- To delete the all the documents of the table:

```
db.studentData.deleteMany({})
```

- To group specific field together:
  (This will display the email and _id)

```
db.studentData.find({}, {email : 1})
```

- To group specific field together:
  (_id by default is 1)

```
db.studentData.find({name : "Harry"}, {email : 1, _id : 0, name : 1})
```

- Converting the group of data into an Array:

```
db.studentData.find({name : "Harry"}, {email : 1, _id : 0}).toArray()
```

- To get a particular data from a document:

```
db.studentData.findOne({name : "Adesh"}).email
db.studentData.findOne({name : "Adesh"}).age
db.studentData.findOne({name : "Adesh"}).hobby[1]
```

- How to store a data into a variable:

```
var studentid = db.studentData.findOne({name : "Travour"})._id
studentid
```

- How to fetch data from the variable (studentid):

```
db.studentData.find(_id : studentid)
db.studentData.findOne(_id : studentid).email
```

- How to print all the documents using ForEach Loop
  (If in your collection there is more than 20 documents then '**.find()**' function will display only the first 20 documents and to display the next 20 documents you have type 'it' to display more So, by using for each loop you will see all the documents present in the collections.)

```
db.studentData.find().forEach((student) => {printjson(student)})
```

- How to print specific field from all the documents:

```
db.studentData.find().forEach((student) =>
```