Arduino IDE Cheatsheet

A quick reference for the Arduino Integrated Development Environment, covering essential keyboard shortcuts, navigation, coding tips, and workflow essentials.

Basic Operations & Editing

File & Navigation Shortcuts

New Sketch	`Ctrl+N` / `Cmd+N`
Open Sketch	`Ctrl+O` / `Cmd+O`
Save Sketch	`Ctrl+S` / `Cmd+S`
Save Sketch As...	`Ctrl+Shift+S` / `Cmd+Shift+S`
Close Window	`Ctrl+W` / `Cmd+W`
Quit IDE	`Ctrl+Q` / `Cmd+Q`
Next Tab	`Ctrl+PageDown` / `Cmd+PageDown`
Previous Tab	`Ctrl+PageUp` / `Cmd+PageUp`

Code Editing Shortcuts

Cut	`Ctrl+X` / `Cmd+X`
Copy	`Ctrl+C` / `Cmd+C`
Paste	`Ctrl+V` / `Cmd+V`
Undo	`Ctrl+Z` / `Cmd+Z`
Redo	`Ctrl+Y` / `Cmd+Shift+Z`
Select All	`Ctrl+A` / `Cmd+A`
Find	`Ctrl+F` / `Cmd+F`
Find Next	`Ctrl+G` / `Cmd+G`
Find Previous	`Ctrl+Shift+G` / `Cmd+Shift+G`

Sketch Structure

Every Arduino sketch must contain two functions:

- `setup()`: This function runs once when the sketch starts after powering up or resetting the board. It's used for initializing variables, pin modes, serial communication, and including libraries.

```
void setup() {
  // Initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}
```

- `loop()`: This function runs continuously after the `setup()` function finishes. It's where the main logic of your program resides, handling tasks like reading sensors, controlling actuators, and communication.

```
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off
  delay(1000);                       // wait for a second
}
```

Comments:

Use comments to explain your code. They are ignored by the compiler.

- Single-line comment:

```
// This is a single-line comment
```

- Multi-line comment:

```
/*
 * This is a multi-line comment
 * spanning multiple lines.
 */
```

Variable Declaration:

Variables are declared before they are used, typically at the beginning of `setup()`, `loop()`, or globally at the top of the sketch.

```
int sensorValue; // declare an integer variable
const int ledPin = 13; // declare a constant integer
```

Include Libraries:

Libraries provide extra functionality. Include them at the very top of the sketch.

```
#include <Wire.h> // Include Wire library for I2C communication
```

Datatypes:

Common datatypes include:

- `int`: Integers (-32,768 to 32,767)
- `float`: Floating-point numbers
- `boolean`: `true` or `false`
- `char`: Characters
- `byte`: 8-bit unsigned numbers (0 to 255)
- `String`: Text strings
- `void`: Indicates no value

Functions:

User-defined functions can be created to organize code. Define them outside of `setup()` and `loop()`.

```
void myFunction(int parameter) {
  // code goes here
}
```

Compilation & Upload

Verify & Upload Actions

Verify Sketch	`Ctrl+R` / `Cmd+R` (Checks code for errors, doesn't upload)
Upload Sketch	`Ctrl+U` / `Cmd+U` (Compiles and uploads to selected board)
Upload Using Programmer	`Ctrl+Shift+U` / `Cmd+Shift+U` (For bootloader burning or specific programmers)
Show Sketch Folder	`Ctrl+K` / `Cmd+K` (Opens the folder containing the current sketch file)
Show Verbose Output during Compilation/Upload	File -> Preferences -> Show verbose output (Useful for debugging issues)
Export Compiled Binary	Sketch -> Export Compiled Binary (Saves the .hex file for direct upload)
Burn Bootloader	Tools -> Burn Bootloader (Requires a programmer and correct fuse settings)

Error Handling Tips

Read the Output: The black console area at the bottom provides error messages. The first error is often the root cause.
Syntax Errors: Look for missing semicolons (`;`), incorrect parentheses `()`, curly braces `{}` or square brackets `[]`.
Compiler Errors: Messages like "'symbol' was not declared in this scope" usually mean you misspelled a variable/function name, or it's out of scope, or you forgot to `#include` a library.
Linker Errors: Often relate to libraries or functions that aren't correctly implemented or found. Check library installation and includes.
Upload Errors: Problems communicating with the board. Ensure the correct board and port are selected (Tools menu). Check USB cable, board connection, and bootloader state. Sometimes a double-tap on the reset button helps.
Common Typos: `digitalWrite(pin, HIGH)` vs `digitalRead(pin)`, `Serial.begin(rate)` vs `Serial.print()`, forgetting `pinMode()`.
Use Auto Format: `Ctrl+T` / `Cmd+T` can help catch syntax issues by formatting your code consistently.
Google the Error: Copy and paste the exact error message into a search engine. Chances are, someone else has had the same problem.

Board & Port Selection

Select Board	Tools -> Board -> [Select your board] (e.g., 'Arduino Uno', 'ESP32 Dev Module')
Select Port	Tools -> Port -> [Select your COM or /dev/tty port] (This is the USB connection to your board)
Install Board Packages	Tools -> Board -> Boards Manager... (Search and install definitions for boards not included by default, like ESP8266/ESP32)
Install Library	Sketch -> Include Library -> Manage Libraries... (Search and install common libraries)
Add .ZIP Library	Sketch -> Include Library -> Add .ZIP Library... (For manually downloaded libraries)
Check Board Info	Tools -> Get Board Info (Displays basic info about the connected board)
Programmer Selection	Tools -> Programmer -> [Select your programmer type] (Usually needed only for bootloader burning or specific chips)

Tools, Libraries & Tips

Serial Monitor & Plotter

Open Serial Monitor	`Ctrl+Shift+M` / `Cmd+Shift+M` (Used for text communication with the board)
Open Serial Plotter	Tools -> Serial Plotter (Used for plotting comma-separated numerical data)
Set Baud Rate	Dropdown menu in Serial Monitor/Plotter (Must match `Serial.begin()` rate in code)
Send Data (Serial Monitor)	Type in input box, hit Enter or Send button (Newline options at bottom)
Clear Output (Serial Monitor)	Click 'Clear output' button
Autoscroll (Serial Monitor)	Checkbox 'Autoscroll'

Example `Serial.print`	`Serial.begin(9600); // Initialize serial comms`
	`Serial.println("Hello, World!"); // Print text with newline`
	`Serial.print(sensorValue); // Print value`

Library Management

Using the Library Manager:

1. Go to `Sketch -> Include Library -> Manage Libraries...`
2. Search for the library you need (e.g., `Adafruit_Unified_Sensor`).
3. Click on the library and select 'Install'.
4. Restart the IDE if prompted.

Including in Sketch:

After installing, include the library at the top of your sketch using `#include`.

```
#include <MySensorLibrary.h>
```

Adding a .ZIP Library:

1. Download the library as a `.zip` file.
2. Go to `Sketch -> Include Library -> Add .ZIP Library...`
3. Navigate to and select the downloaded `.zip` file.
4. The IDE will install it. Restart if necessary.

Finding Examples:

Installed libraries often come with examples.

Go to `File -> Examples -> [Library Name] -> [Example Sketch]`

Where Libraries are Stored:

Usually in your Arduino sketchbook folder (`Documents/Arduino` by default).

- Contributed libraries are in `sketchbook/libraries/`.
- Built-in libraries are in the IDE installation folder.

Updating Libraries:

The Library Manager will show if updates are available for installed libraries.

Troubleshooting Libraries:

Ensure the library is compatible with your board. Check the library documentation or GitHub page for requirements.

Tips & Advanced Features

Coding & Debugging Tips

Use Serial.print() for Debugging: Print variable values, messages, or states to the Serial Monitor to understand code execution flow.

Define Constants: Use `const int` or `#define` for pin numbers and fixed values. Makes code more readable and easier to modify.

Use `long` for Time: When working with `millis()`, use `unsigned long` variables to avoid overflow issues after ~50 days.

Avoid `delay()` in Complex Sketches: `delay()` halts the entire program. Use the `millis()` function to manage timing without blocking.

Name Pins Clearly: Give pins descriptive names using constants (e.g., `const int ledPin = 13;`).

Break Down Code into Functions: Organize reusable blocks of code into separate functions to improve readability and maintainability.

Check Examples: The built-in examples (`File -> Examples`) are a great resource for learning how to use different features and libraries.

Use `F()` Macro for Strings: When printing constant strings to Serial, use `Serial.println(F("My String"));` to store them in flash memory instead of RAM, saving precious RAM.

IDE Preferences

Access Preferences: `File -> Preferences` (Windows/Linux) or `Arduino -> Preferences` (macOS).

Sketchbook Location: Change the default folder for your sketches and libraries.

Editor Language: Change the language of the IDE interface.

Editor Font Size: Adjust text size in the code editor.

Show Verbose Output: Enable detailed output during compilation and upload (mentioned before, useful for debugging).

Compiler Warnings: Set warning levels (e.g., 'More warnings') to catch potential issues in your code.

Check for Updates: Configure whether the IDE checks for software updates automatically.

Useful Menu Items

Import .pde Sketch	File -> Import .pde Sketch (For older sketch formats)
Examples	File -> Examples -> [Built-in, Libraries] (Starter code for various functionalities)
Manage Libraries	Sketch -> Include Library -> Manage Libraries... (Find, install, and update libraries)
Manage Boards	Tools -> Board -> Boards Manager... (Install board definitions for different microcontrollers)
Auto Format	Tools -> Auto Format (Cleans up code indentation and spacing)
Archive Sketch	Sketch -> Archive Sketch (Creates a `.zip` file of your sketch folder)
Serial Monitor	Tools -> Serial Monitor (Text communication console)
Serial Plotter	Tools -> Serial Plotter (Graphical data plotter)