🤖 Student API Cheatsheet

This page explains the **tools your bot can use** during a battle.
You will use these tools inside a function called `tick(api)`.

Think of `api` as **your bot's senses and controls**.
Each turn of the battle, your code decides what the bot should do next.

🧠 The `tick(api)` Function

What it is

`tick(api)` runs **once every game step** (called a *tick*).
This is where you write your bot's decision-making code.

Why this matters

If something isn't inside `tick(api)`, your bot **can't do it** during the game.

Basic example

```
function tick(api) {
      api.turn(1);
      api.fire();
}
```

What's happening here:

- The bot turns (just a bit, clockwise)
- It fires its laser

This example will work a rotating automatic laser turret. Try it!. You can do so by replacing the content of your quickstart bot's behavior.js file. Pretty cool, eh? You might not win too many battles with this, but it's better than just `advance()`ing forward into the wall…

What Your Bot Can Do (and How)

The rest of this document lists **everything your bot is allowed to do** inside the arena.

You do **not** need to memorize this.
Use it like a menu or toolbox while you experiment.

🧠 How to Read This Document

- Every command is something you can use **inside `tick(api)`**
- Your bot can issue engage in limited actions per tick
    - One scan
    - One fire
    - One turn
    - One advance

Example context:

```
function tick(api) {
  // Your code goes here
}
```

🔁 The Game Loop (Quick Reminder)

Every tick:

1. The game calls `tick(api)`
2. Your code runs once
3. The bot acts
4. The next tick begins

Your bot **does not remember anything** unless you store it on purpose.

👀 Seeing the World

`api.scan(fovDegrees (OPTIONAL))`

Looks for the nearest enemy in front of your bot.

Example:

```
let result = api.scan();
let result = api.scan(90);
```

What it returns:

```
{
      found: true or false,
      angle: number,      // degrees to turn toward the enemy
      distance: number    // how far away the enemy is
      id: String          // the found bot's name
}
```

Accessing the results uses dot notation:

```
result.found
result.angle
result.distance
result.id
```

Notes:

- `fovDegrees` is the width of vision (like a flashlight beam)
    - If you don't use fovDegrees, your bot will scan its maximum field possible
    - Your bot's max field depends on your build configuration
- If `found` is `false`, no enemy was seen
- You can only scan once per tick

🔄 Turning

`api.turn(degrees)`

Rotates your bot.

Examples:

```
api.turn(5);     // turn right
api.turn(-5);    // turn left
```

Tips:

- Small turns are smoother
- Turning does not move your bot forward or backward. It just rotates, or spins in place.
- Turning too fast can cause overshooting

🚶 Moving

`api.advance(amount)`

Moves your bot forward.

Example:

```
api.advance(0.5);
```

Notes:

- Values for numbers in the brackets are between `0` and `1`
    - Negative values not allowed (no reversing)
- The speed your robot actually moves depends on your build

🔫 Shooting

`api.fire()`

Fires a laser shot straight ahead.

Example:

```
api.fire();
```

Important:

- Shots are **instant** (they have no travel time)
- If you're not facing the enemy, the shot will miss.
- Part of your score calculation considers your bot's accuracy
- Your laser's range depends on your build
- Your bot can fire only once per tick

🎯 Aiming Help

`api.aligned(scanResult, tolerance (OPTIONAL))`

Checks if your bot is aiming close enough to shoot, meaning it falls within the tolerance angle given. If you leave tolerance out, which is probably easiest, especially to start, it will default to 6 degrees.

Example:

```
let acquiredTarget = api.scan();
if (api.aligned(acquiredTarget)) {
  api.fire();
}
```

Why this matters:

- Prevents wasting shots
- Helps with precise bots
- `angleTolerance` is in degrees

🧠 Remembering Things

Bots forget everything by default.
You can give them memory.

`api.memorySet(memoryNum, storedValue)`

Stores a number in one of your bot's memory locations. There is a maximum memory size of five items. Locations are numbered 0-4.

Example:

```
api.memorySet(0, result.distance);
```

`api.memoryGet(memoryNum)`

Retrieves stored data.

Example:

```
let last = api.memoryGet(0);
```

Notes:

- Memory only stores numbers
- Memory size depends on your build configuration
    - Default is no memory at all!
- Use memory only after your bot works without it

📊 Knowing Yourself

`api.getState()`

Returns information about your own bot.

Example:

```
let state = api.getState();
```

Accessible info:

`State.health` - remaining HP of bot
`state.x` - horizontal coordinate of bot position
`state.y` - vertical coordinate of bot position
`state.heading` - a number in degrees (0 - right, 90 - down, 180 - left, 270 - up)
`state.time` - how much match time has elapsed
`state.alive` - true when alive, false after being destroyed

Use cases:

- Retreat when health is low
- Detect if you're stuck
- Smarter positioning

🚧 Walls and Collisions

Your bot cannot leave the arena.

Wall behavior depends on your build:

- `"stop"` → bot stops at walls
- `"bounce"` → bot reflects
- `"slide"` → bot slides along the wall

You do not control walls directly - plan around them.

❗ Common Beginner Mistakes

❌ Forgetting to scan

Your bot can't react if it never looks.

❌ Firing without aiming

Use `turn()` or `aligned()` first.

❌ Doing nothing

A bot that never moves, turns, or fires is technically valid - but not useful.

❌ Changing too much at once

Small changes = easier debugging.

🧠 Prioritize Your Understanding of The API

In the beginning, you do **not** need to master:

- Memory strategies
- Perfect aiming
- Scoring optimization
- Advanced movement tricks

Those can come with experience.

✅ Final Advice

You are designing **behavior**.

Your bot is not reacting to a remote control. At any point, all it can do is:

- Acquire information about the match state
- Make decisions to (re)act accordingly

As you progress through more battles, and add more complex behaviors, if you can answer:

> "Why did my bot do that?"

…you're learning exactly what BotBattles is meant to teach.