# Cheat Sheet for GSAP + Glaze + WordPress + BricksBuilder □□

This cheat sheet is designed as your **practical quick-reference companion** for future lessons. It is **not yet the full course**, but a structured reference you can keep beside you while building animations.

It assumes:

1. You use **WordPress**
2. You build with **BricksBuilder**
3. You want to learn **GSAP properly**
4. You also want to benefit from **Glaze for simpler workflows**
5. Your **JavaScript and CSS knowledge is still developing**

---

# 1. Big Picture: What Each Tool Does

1. **CSS**
    1. Controls styling and layout
    2. Can do basic transitions and animations
    3. Is often enough for simple hover effects
    4. Becomes limited for complex sequences and scroll behavior
2. **JavaScript**
    1. Lets you control what happens in the browser
    2. Can respond to events such as:
        1. click
        2. scroll
        3. page load
    3. Makes animation logic possible
3. **GSAP**

1. A JavaScript animation library
2. Makes animations smoother and easier to control
3. Excellent for:
    1. sequencing
    2. stagger effects
    3. scroll animations
    4. advanced motion systems
    5. reusable animation logic
4. **ScrollTrigger**
    1. A GSAP plugin
    2. Connects animation behavior to scroll position
    3. Used for:
        1. reveal-on-scroll
        2. scrub effects
        3. pinned sections
        4. storytelling layouts
5. **Glaze**
    1. A simplification layer for animation workflows
    2. Helps you declare animations more quickly
    3. Useful when you want:
        1. less custom JavaScript
        2. faster implementation
        3. repeatable simple effects
    4. Less ideal when you need:
        1. very custom logic
        2. complex sequencing
        3. heavy interactivity

---

# 2. Core Mental Model of GSAP □□

Think of GSAP like this:

1. **Target**
    1. What element should animate?
2. **Property**
    1. What should change?
    2. Examples:
        1. `x`
        2. `y`
        3. `opacity`
        4. `scale`
        5. `rotation`
3. **Value**

    1. How much should it change?
4. **Timing**
    1. How long should it take?
    2. Should it start later?
    3. Should it ease smoothly?

Basic pattern:

```
gsap.to(".box", {
  x: 100,
  opacity: 1,
  duration: 1,
  ease: "power2.out"
});
```

Meaning:

1. Animate `.box`
2. Move it `100` pixels on the x-axis
3. Make it fully visible
4. Take `1` second
5. Use a smooth easing curve

---

# 3. The 3 Most Important GSAP Methods

## 3.1 `gsap.to()`

Use when you want to animate **from the current state to a new state**.

```
gsap.to(".box", {
  y: -50,
  opacity: 1,
  duration: 1
});
```

**Meaning:**
The element starts from however it currently looks and animates to these values.

## 3.2 `gsap.from()`

Use when you want to animate **from a starting state into the natural/current state**.

```
gsap.from(".box", {
  y: 50,
  opacity: 0,
  duration: 1
});
```

**Meaning:**
The element appears to come in from below and fade in.

This is extremely common for **entrance animations**.

---

## 3.3 `gsap.fromTo()`

Use when you want to define **both start and end values explicitly**.

```
gsap.fromTo(".box",
  {
    y: 50,
    opacity: 0
  },
  {
    y: 0,
    opacity: 1,
    duration: 1
  }
);
```

**Use this when:**

1. You want maximum clarity
2. You do not want to depend on the element's existing CSS state
3. You want more predictable results

---

# 4. Most Important GSAP Properties

## 4.1 Movement

1. `x`
    1. Moves left or right
    2. Positive = right
    3. Negative = left

```
x: 100
```

2. `y`
    1. Moves up or down
    2. Positive = down
    3. Negative = up

```
y: -50
```

---

## 4.2 Visibility

1. `opacity`
    1. `0` = invisible
    2. `1` = fully visible

```
opacity: 0
```

---

## 4.3 Size

1. `scale`
    1. `1` = normal size
    2. `0.8` = smaller
    3. `1.2` = larger

```
scale: 0.8
```

---

# 4.4 Rotation

1. `rotation`
    1. Rotates in degrees

```
rotation: 45
```

---

# 4.5 Timing

1. `duration`
    1. How long animation takes in seconds

```
duration: 1.2
```

2. `delay`
    1. Wait before starting

```
delay: 0.3
```

3. `ease`
    1. Controls animation feel

```
ease: "power2.out"
```

---

# 5. Most Common Ease Values

Use these often:

1. `linear`
    1. Constant speed
    2. Often feels mechanical
2. `power1.out`
    1. Gentle ease out
    2. Good for subtle motion
3. `power2.out`
    1. Very common default feel
    2. Smooth and professional
4. `power3.out`

    1. Stronger easing
    2. Feels more dramatic
5. `power2.inOut`
    1. Smooth at start and end
    2. Good for balanced movement
6. `back.out`
    1. Slight overshoot
    2. Good for playful emphasis
7. `elastic.out`
    1. Spring-like
    2. Use carefully

**Simple recommendation:**
For many professional websites, start with:

```
ease: "power2.out"
```

---

# 6. Targets and Selectors

GSAP needs to know **which element** to animate.

## 6.1 Common selector types

1. **Class selector**

```
".card"
```

2. **ID selector**

```
"#hero-title"
```

3. **Nested selector**

```
".hero .heading"
```

4. **Multiple matching elements**

```
".feature-card"
```

If many elements have that class, GSAP can animate all of them.

# 6.2 Good selector habits in BricksBuilder 

1. Prefer **classes** over IDs in most cases
2. Use **stable custom classes** you control
3. Avoid overly long selectors when possible
4. If content repeats, make sure your selector strategy still makes sense

Good examples:

```
".js-fade-up"
".hero-title"
".feature-card"
".testimonial-item"
```

Better to use purposeful classes such as:

1. `.js-reveal`
2. `.js-stagger-item`
3. `.js-hero-title`

This makes your code easier to reuse.

---

# 7. `gsap.set()` for Initial States

Use `gsap.set()` when you want to assign values immediately without animation.

```
gsap.set(".box", {
  opacity: 0,
  y: 40
});
```

Then animate later:

```
gsap.to(".box", {
  opacity: 1,
  y: 0,
```

```
  duration: 1
});
```

This is useful for:

1. predictable starting states
2. avoiding flashes of visible content
3. cleaner animation setup

---

# 8. Stagger Animations

A **stagger** means multiple elements animate one after another.

```
gsap.from(".card", {
  y: 30,
  opacity: 0,
  duration: 0.8,
  stagger: 0.15
});
```

Meaning:

1. Each `.card` animates in
2. A `0.15` second gap exists between starts

Very useful for:

1. card grids
2. lists
3. icon rows
4. gallery items
5. repeated Bricks query loop items

---

# 9. Timelines: The Most Important Intermediate Concept □

If you learn only one big GSAP concept beyond basics, learn **timelines**.

# 9.1 Why timelines matter

Without timelines, you often write many separate animations.

With timelines, you can control a sequence in one place.

---

# 9.2 Basic timeline example

```
let tl = gsap.timeline();

tl.from(".hero-badge", {
  y: 20,
  opacity: 0,
  duration: 0.6
})
.from(".hero-title", {
  y: 40,
  opacity: 0,
  duration: 0.8
})
.from(".hero-text", {
  y: 20,
  opacity: 0,
  duration: 0.6
})
.from(".hero-button", {
  y: 20,
  opacity: 0,
  duration: 0.5
});
```

This creates a clean sequence.

---

# 9.3 Overlapping timeline animations

```
let tl = gsap.timeline();

tl.from(".hero-title", {
  y: 40,
  opacity: 0,
  duration: 0.8
})
.from(".hero-text", {
  y: 20,
  opacity: 0,
  duration: 0.6
}, "-=0.3");
```

`"-=0.3"` means:

1. Start this animation `0.3` seconds before the previous one ends
2. This creates overlap
3. Overlap often feels more polished

---

# 10. ScrollTrigger Cheat Sheet □□

## 10.1 Basic setup

```
gsap.registerPlugin(ScrollTrigger);
```

Then:

```
gsap.from(".section-title", {
  y: 40,
  opacity: 0,
  duration: 0.8,
  scrollTrigger: {
    trigger: ".section-title",
    start: "top 80%"
  }
});
```

---

# 10.2 Most important ScrollTrigger options

1. `trigger`
    1. Which element controls when the animation starts
2. `start`
    1. When the trigger activates

Example:

```
start: "top 80%"
```

Meaning:

1. when the top of the trigger reaches `80%` down the viewport
2. `end`
    1. Where the trigger effect ends
    2. More relevant for scrub or pin setups
3. `scrub`
    1. Links animation progress to scroll progress

```
scrub: true
```

5. `pin`
    1. Keeps an element fixed during scroll for a section of time

```
pin: true
```

6. `markers`
    1. Shows visual debug markers on screen

```
markers: true
```

Very useful when learning.

7. `toggleActions`
    1. Controls what happens on enter/leave/re-enter/back

Common example:

```
toggleActions: "play none none none"
```

---

# 10.3 Simple reveal on scroll

```
gsap.from(".reveal", {
  y: 40,
  opacity: 0,
  duration: 0.8,
  ease: "power2.out",
  scrollTrigger: {
    trigger: ".reveal",
    start: "top 85%"
  }
});
```

# 10.4 Stagger reveal on scroll

```
gsap.from(".card", {
  y: 30,
  opacity: 0,
  duration: 0.8,
  stagger: 0.15,
  scrollTrigger: {
    trigger: ".cards-wrapper",
    start: "top 80%"
  }
});
```

# 10.5 Scrub example

```
gsap.to(".image", {
  y: -100,
  ease: "none",
  scrollTrigger: {
    trigger: ".section",
    start: "top bottom",
    end: "bottom top",
```

```
    scrub: true
  }
});
```

This creates a scroll-linked effect.

---

# 11. Common Animation Patterns You Will Use Often

## 11.1 Fade in

```
gsap.from(".item", {
  opacity: 0,
  duration: 0.8
});
```

---

## 11.2 Fade up

```
gsap.from(".item", {
  y: 30,
  opacity: 0,
  duration: 0.8,
  ease: "power2.out"
});
```

---

## 11.3 Slide in from left

```
gsap.from(".item", {
  x: -60,
  opacity: 0,
  duration: 0.8
```

```
});
```

# 11.4 Zoom in

```
gsap.from(".item", {
  scale: 0.85,
  opacity: 0,
  duration: 0.8
});
```

# 11.5 Sequential hero animation

```
let tl = gsap.timeline();

tl.from(".hero-kicker", {
  y: 20,
  opacity: 0,
  duration: 0.5
})
.from(".hero-title", {
  y: 40,
  opacity: 0,
  duration: 0.8
}, "-=0.2")
.from(".hero-text", {
  y: 20,
  opacity: 0,
  duration: 0.6
}, "-=0.3")
.from(".hero-button", {
  y: 20,
  opacity: 0,
  duration: 0.5
}, "-=0.2");
```

# 12. CSS Concepts You Must Know for Animation

You do **not** need advanced CSS first, but you do need these basics.

## 12.1 `transform`

GSAP often animates transforms.

Examples:

1. `translateX`
2. `translateY`
3. `scale`
4. `rotate`

GSAP simplifies this with properties like:

1. `x`
2. `y`
3. `scale`
4. `rotation`

---

## 12.2 `opacity`

Controls visibility.

```
opacity: 0;
```

Invisible, but still takes up layout space.

---

## 12.3 `position`

Important values:

1. **`static`**
    1. default

2. **`relative`**
    1. keeps normal flow
    2. allows positioned children to reference it
3. **`absolute`**
    1. taken out of normal flow
    2. often used for layered animation elements
4. **`fixed`**
    1. fixed to viewport

---

# 12.4 `overflow`

Very important for reveals.

```
overflow: hidden;
```

Useful when:

1. text slides upward inside a wrapper
2. images reveal from inside a frame
3. you want to hide animated overflow

---

# 12.5 `z-index`

Controls stacking order.

If animation looks wrong, sometimes the issue is not GSAP but layering.

---

# 12.6 `display`

Common values:

1. `block`
2. `inline`
3. `inline-block`
4. `flex`
5. `grid`
6. `none`

Animation can behave differently depending on display type.

# 13. JavaScript Basics You Need Most

## 13.1 Variables

```
let tl = gsap.timeline();
```

A variable stores something.

---

## 13.2 Functions

```
function runAnimation() {
  gsap.from(".box", {
    y: 30,
    opacity: 0
  });
}
```

A function groups code for reuse.

---

## 13.3 Selecting elements

```
document.querySelector(".box");
document.querySelectorAll(".card");
```

GSAP often lets you skip this and pass selectors directly.

---

## 13.4 Run code after page load

Common pattern:

```
document.addEventListener("DOMContentLoaded", () => {
  gsap.from(".box", {
    y: 30,
    opacity: 0
  });
});
```

Useful in WordPress/Bricks to avoid running too early.

---

# 14. WordPress + BricksBuilder Implementation Cheat Sheet □□

## 14.1 Good places for custom animation code

Depending on your setup, you may place code in:

1. Bricks custom code area
2. page-level custom code
3. theme or child-theme files
4. a code snippets plugin
5. footer script area

For beginners, a **safe custom code area** is usually easiest.

---

## 14.2 Recommended structure

1. Add GSAP library
2. Add ScrollTrigger if needed
3. Add your own custom script after GSAP
4. Give your Bricks elements meaningful classes

---

# 14.3 Example HTML class strategy in Bricks

For a hero section, assign classes such as:

1. `.hero-section`
2. `.hero-title`
3. `.hero-text`
4. `.hero-button`

Then target them in GSAP.

---

# 14.4 Good naming style

Use animation-friendly class names:

1. `.js-fade-up`
2. `.js-stagger-group`
3. `.js-stagger-item`
4. `.js-hero-title`

This makes your intent clear.

---

# 14.5 Query loop caution

If Bricks repeats items, selectors may target **all repeated items**.

That is good when intended, but dangerous if you expected only one.

Be careful with:

1. repeated cards
2. post lists
3. product loops
4. sliders

---

# 15. Debugging Cheat Sheet □□

When animation fails, check in this order:

1. **Is GSAP loaded?**
    1. Open console
    2. Test:

```
console.log(gsap);
```

2. **Is ScrollTrigger loaded and registered?**
    1. Test:

```
console.log(ScrollTrigger);
```

3. **Does the selector match a real element?**
    1. Test:

```
console.log(document.querySelector(".my-element"));
```

4. **Is the script running too early?**
    1. Wrap in `DOMContentLoaded`
5. **Is CSS preventing the result?**
    1. Hidden parent
    2. `overflow`
    3. `display: none`
    4. wrong positioning
    5. transform conflict
6. **For scroll animation: is the trigger correct?**
    1. Use `markers: true`
7. **Are there JavaScript errors above your GSAP code?**
    1. One earlier error can stop everything

---

# 16. Common Beginner Mistakes

1. **Using the wrong selector**
    1. You typed `.hero-title`
    2. The real class is `.hero_heading`
2. **Trying to animate before the element exists**
    1. Common in builders or dynamic content
3. **Confusing `from()` and `to()`**
    1. `from()` = animate from values into current state
    2. `to()` = animate from current state to new values
4. **Animating layout properties instead of transforms**

    1. Avoid animating things like:
        1. `top`
        2. `left`
        3. `width`
        4. `height`
    2. Prefer:
        1. `x`
        2. `y`
        3. `scale`
        4. `opacity`
5. **Using too much animation**
    1. A professional website usually needs restraint
6. **Not testing mobile**
    1. Motion distance may feel too large on small screens
7. **Ignoring reduced motion**
    1. Accessibility matters

---

# 17. Performance Cheat Sheet ⚡

## Prefer these properties

1. `x`
2. `y`
3. `scale`
4. `rotation`
5. `opacity`

## Be careful with these

1. `width`
2. `height`
3. `top`
4. `left`
5. box-shadow-heavy animation
6. filter-heavy animation

## General rules

1. Keep animations purposeful

2. Do not animate everything on a page
3. Avoid too many simultaneous scroll effects
4. Test on mobile
5. Use stagger for elegance instead of chaos

---

# 18. Accessibility Cheat Sheet ♿

## Respect reduced motion

Basic idea:

1. Some users prefer less motion
2. Your site should reduce or remove intense animation for them

Conceptually, your animations should:

1. avoid large aggressive movement
2. avoid constant motion
3. avoid distracting flashing behavior
4. support content, not fight it

A simple CSS concept you should know:

```
@media (prefers-reduced-motion: reduce) {
  * {
    animation: none !important;
    transition: none !important;
  }
}
```

Later in the course, we can build the **GSAP version** of this properly.

---

# 19. GSAP Pattern Templates

## 19.1 Basic entrance animation

```
gsap.from(".element", {
  y: 30,
  opacity: 0,
  duration: 0.8,
  ease: "power2.out"
});
```

# 19.2 Scroll reveal template

```
gsap.from(".element", {
  y: 30,
  opacity: 0,
  duration: 0.8,
  ease: "power2.out",
  scrollTrigger: {
    trigger: ".element",
    start: "top 85%",
    markers: false
  }
});
```

# 19.3 Stagger template

```
gsap.from(".item", {
  y: 20,
  opacity: 0,
  duration: 0.6,
  ease: "power2.out",
  stagger: 0.12
});
```

# 19.4 Hero timeline template

```
let tl = gsap.timeline();

tl.from(".hero-title", {
  y: 40,
  opacity: 0,
  duration: 0.8,
  ease: "power2.out"
})
.from(".hero-text", {
  y: 20,
  opacity: 0,
  duration: 0.6,
  ease: "power2.out"
}, "-=0.3")
.from(".hero-button", {
  y: 20,
  opacity: 0,
  duration: 0.5,
  ease: "power2.out"
}, "-=0.2");
```

---

# 20. Glaze Cheat Sheet □□

Because Glaze setups can vary depending on version and implementation style, I'll keep this section **conceptually accurate and workflow-oriented** for now, so you do not learn the wrong syntax too early.

## 20.1 What Glaze is good for

1. Simple reveal effects
2. Repeatable animation rules
3. Lower-code workflows
4. Faster implementation in visual-builder environments

---

## 20.2 Glaze mental model

Instead of writing lots of custom GSAP code, you often:

1. add attributes or configuration to elements
2. declare animation behavior in a simpler way
3. let Glaze translate that into GSAP behavior

---

# 20.3 Best use cases for Glaze

1. fade-up reveals
2. simple scroll reveals
3. repeated animation classes or attributes
4. site-wide animation patterns
5. quick implementation in Bricks

---

# 20.4 When to switch to raw GSAP instead

Use raw GSAP when you need:

1. custom hero timelines
2. overlapping multi-step sequences
3. advanced ScrollTrigger control
4. interactive states
5. custom animation logic
6. timeline labels and playback control

---

# 20.5 Practical rule of thumb

1. **Simple and repeatable** → use **Glaze**
2. **Custom and complex** → use **GSAP**
3. **Mixed project** → use both

That hybrid approach is often ideal.

---

# 21. “Which Tool Should I Use?” Decision Guide

1. **I want a simple fade-up on scroll for many sections**
    1. Best choice: **Glaze** or simple GSAP
2. **I want a premium hero animation with perfect sequence control**
    1. Best choice: **GSAP timeline**
3. **I want cards to enter one after another**
    1. Best choice: **GSAP stagger**
    2. Or Glaze if repeated setup is simple enough
4. **I want animation tied to scroll progress**
    1. Best choice: **GSAP + ScrollTrigger**
5. **I want low-code convenience**
    1. Best choice: **Glaze**
6. **I want to deeply understand what is happening**
    1. Best choice: learn **GSAP first**

---

# 22. Professional Defaults You Can Reuse

Here are safe defaults that often work well on professional websites:

## 22.1 Standard fade-up

```
{
  y: 30,
  opacity: 0,
  duration: 0.8,
  ease: "power2.out"
}
```

---

## 22.2 Standard stagger

```
{
  y: 20,
  opacity: 0,
  duration: 0.6,
  ease: "power2.out",
```

```
  stagger: 0.12
}
```

---

# 22.3 Standard scroll trigger

```
{
  trigger: ".element",
  start: "top 85%"
}
```

---

# 22.4 Good motion design advice

1. Use small to medium distances
2. Use smooth easing
3. Keep durations moderate
4. Avoid exaggerated movement unless stylistically justified

---

# 23. Quick Glossary

1. **Target**
   1. The element being animated
2. **Tween**
   1. A single GSAP animation
3. **Timeline**
   1. A sequence of animations
4. **Stagger**
   1. A delayed sequence across multiple elements
5. **Ease**
   1. The speed curve of motion
6. **Trigger**
   1. The element that controls when a scroll animation starts
7. **Scrub**
   1. Scroll directly controls animation progress
8. **Pin**
   1. Element stays fixed during part of scroll
9. **Transform**
   1. Visual movement/scale/rotation without changing layout flow

10. **Opacity**
    1. Visibility level

---

# 24. Your Personal Beginner Workflow

When creating an animation, think in this order:

1. **What element am I animating?**
2. **What should visually change?**
3. **Should this happen on page load or on scroll?**
4. **Do I need one animation or a sequence?**
5. **Can I use a simple repeated pattern?**
    1. If yes, consider **Glaze**
6. **Do I need detailed control?**
    1. If yes, use **GSAP**
7. **Does it still work well on mobile?**
8. **Is it subtle enough to feel professional?**

---

# 25. Super Short “Memory Version” 

If you remember only a few things, remember these:

1. **Use `from()` for entrance animations**
2. **Use `to()` for animating to a new state**
3. **Use timelines for sequences**
4. **Use ScrollTrigger for scroll behavior**
5. **Use transforms and opacity for best performance**
6. **Use stable classes in Bricks**
7. **Use Glaze for simple repeated patterns**
8. **Use raw GSAP for complex custom motion**
9. **Debug with selectors, load order, and markers**
10. **Subtle animation usually looks more professional**

---

# 26. Suggested Next Step □□

A good next step would be one of these:

1. **Create an expanded GSAP-only cheat sheet**
    1. with more code examples
    2. with beginner explanations line by line
2. **Create a BricksBuilder implementation cheat sheet**
    1. exactly where to place scripts
    2. how to structure classes
    3. common builder pitfalls
3. **Create a Glaze-specific cheat sheet**
    1. focused on practical usage patterns
    2. focused on simple animations in Bricks
4. **Start the actual course with Module 1 or Module 2**

If you want, I can next create **three separate cheat sheets**:

1. **GSAP Cheat Sheet**
2. **ScrollTrigger Cheat Sheet**
3. **Glaze + BricksBuilder Cheat Sheet**

That would probably be the most useful version for real practice.

---

Revision #1
Created 2026-04-25 14:16:02 UTC by art10m
Updated 2026-04-25 14:19:07 UTC by art10m