2.3 Core visual concepts

Before you learn **GSAP syntax**, it’s extremely important to understand **what animation is actually changing on screen**. GSAP is “just” the tool. The real foundation is knowing:

1. **what property changes**,
2. **how much it changes**,
3. **when it changes**, and
4. **how those changes relate to other elements**.

If you understand the visual concepts below, GSAP becomes much easier to learn, because you’ll stop thinking in terms of “random code” and start thinking in terms of **motion design decisions** □□

These concepts apply whether you animate with:

- **pure CSS**
- **JavaScript**
- **GSAP**
- **Glaze**
- or even visual builders that output animation code

Why these concepts matter so much

Every animation you have ever seen on a website is usually built from a combination of a few simple ideas:

1. **Position** — where an element is
2. **Scale** — how big or small it is
3. **Rotation** — how much it is turned
4. **Opacity** — how visible it is
5. **Timing** — how fast or slow it moves
6. **Delay** — when it starts
7. **Sequencing** — how multiple animations are arranged together

Even very “advanced” animations are often just clever combinations of these basics.

For example, a fancy hero section reveal might be nothing more than:

1. headline moves upward
2. headline fades in
3. image scales slightly down into place
4. button appears after a short delay
5. everything is timed to feel smooth and intentional

So let’s break each concept down carefully.

2.3.1 Position

What “position” means in animation

Position refers to **where an element appears on the screen**.

When you animate position, you are moving an element from one place to another.

Common directions:

1. **left to right**
2. **right to left**
3. **top to bottom**
4. **bottom to top**
5. **diagonal movement**
6. **movement along a path**

Simple examples

- A button slides in from the left
- A card rises upward when scrolled into view
- A modal drops down from the top
- A gallery image moves slightly on hover
- A floating icon drifts gently up and down

Position in the browser: an important distinction

In web design, position can be changed in multiple ways. This is where beginners often get confused.

There are two major categories:

1. **Layout-based movement**
2. **Transform-based movement**

1. Layout-based movement

This includes changing properties such as:

- `top`
- `left`
- `right`
- `bottom`
- `margin`
- sometimes `width` or `height` in ways that affect layout

These properties often cause the browser to **recalculate layout**, which can be heavier and less performant.

2. Transform-based movement

This includes:

- `transform: translateX(...)`
- `transform: translateY(...)`
- `transform: translate(...)`

This is usually preferred for animation because it is often **smoother and more efficient** □□

In GSAP, this is why you'll frequently animate things like:

- `x`
- `y`

instead of:

- `left`
- `top`

Why transform-based movement is usually better

When you animate with transforms, the browser often doesn’t need to reflow the whole page layout. Instead, it can move the element more efficiently on the rendering layer.

That means:

1. **better performance**
2. **smoother animation**
3. **less risk of layout shifts**
4. **better user experience**

So as a practical rule:

> If you want an element to *appear to move*, prefer **transform movement** over changing layout properties.

Visual intuition for position

Think of an element like a paper card on a table.

- Changing **position** means sliding the card around the table.
- The card itself does not become bigger or smaller.
- It does not become more transparent.
- It simply changes location.

That sounds obvious, but it helps separate position from other concepts like scale and opacity.

Types of positional movement

Horizontal movement

The element moves left or right.

Common use cases:

- off-canvas menus
- carousels
- marquee effects
- text entering from the side

Vertical movement

The element moves up or down.

Common use cases:

- fade-up reveals
- sticky header transitions
- dropdown menus
- “back to top” interactions

Diagonal movement

The element moves across both axes.

This can feel more dynamic, but should be used carefully because it attracts more attention.

Path-based movement

The element follows a more complex route rather than a straight line.

This is more advanced, but still built on the basic idea of changing position over time.

Position and perception

Position affects how users interpret hierarchy and attention.

For example:

1. **movement upward** often feels like appearing or elevating
2. **movement downward** can feel like dropping in or settling
3. **movement from left/right** can feel like entering from outside the viewport
4. **small positional movement** can feel elegant and subtle
5. **large positional movement** can feel dramatic or distracting

A good animation designer asks:

- *How far should this move?*
- *From which direction?*
- *Should the movement be obvious or subtle?*
- *Does the direction support the design?*

Common beginner mistake with position

A very common mistake is moving elements **too far**.

For example:

- text flying in from 500 pixels away
- buttons jumping dramatically
- cards sliding long distances for no reason

Usually, especially in modern websites, **smaller movement looks more professional**.

A subtle move of:

- `10px`
- `20px`
- `30px`
- `50px`

is often enough.

Large movement should be reserved for moments where dramatic motion is intentional.

Position in real website scenarios

Example 1: Hero text reveal

- headline starts slightly lower
- headline moves upward into final place

Why it works:

- feels like the text is settling into position
- creates a polished entry
- combines well with opacity

Example 2: Card hover effect

- card image shifts slightly upward
- icon nudges right
- arrow moves a few pixels

Why it works:

- small movement creates responsiveness
- makes the interface feel interactive
- avoids overwhelming the user

Example 3: Mobile menu

- menu panel slides in from the side

Why it works:

- matches the spatial logic of a hidden panel
- helps users understand where the panel came from

Key principle for position

> Use position animation to guide attention, not to show off motion for its own sake.

If movement helps tell the user *where to look* or *what just happened*, it's useful. If it just creates noise, it's probably too much.

2.3.2 Scale

What "scale" means in animation

Scale means changing the **size** of an element visually.

When an element scales:

- it appears to grow larger, or
- shrink smaller

Importantly, in modern animation this is often done with **transforms**, not by changing actual layout width and height.

For example:

- `scale: 1` means normal size
- `scale: 0.8` means smaller
- `scale: 1.2` means larger

Scale is not the same as width and height

This distinction matters a lot.

Changing width and height

If you animate:

- `width`
- `height`

you often affect layout. Neighboring elements may move, resize, or reflow.

Changing scale

If you animate `transform: scale(...)`, the element visually changes size **without necessarily reflowing the layout**.

That makes scale animation:

1. smoother
2. more efficient
3. better for many interactive effects

Visual intuition for scale

Imagine the same paper card on a table.

- Position = sliding the card
- Scale = making the card appear larger or smaller
- The center point stays roughly in place unless the transform origin changes

---

Why scale is powerful

Scale has strong psychological impact.

A slight increase in scale can suggest:

- importance
- focus
- emphasis
- approachability
- responsiveness

A decrease in scale can suggest:

- moving away
- de-emphasis
- background status
- compression
- exit

This is why scale is used so often for:

- buttons
- cards
- modals
- images
- loading effects
- microinteractions

Common uses of scale

1. Entrance animation

An element starts slightly smaller and scales up into place.

This can make it feel like it is arriving gently.

2. Hover interaction

A button or card scales up slightly on hover.

This gives immediate feedback that the element is interactive.

3. Exit animation

An element scales down as it disappears.

This can make the exit feel cleaner than simply vanishing.

4. Focus effect

A central object scales slightly larger than background objects.

This draws the eye.

Small scale changes are usually best

A common beginner mistake is using scale values that are too extreme.

For professional UI animation, subtle values often work best:

- `0.95`
- `0.98`
- `1.02`
- `1.05`

These feel polished.

Very large changes like:

- `0.2`
- `2`
- `3`

are usually for special effects, not normal interface animation.

Uniform vs non-uniform scaling

Uniform scaling

The element scales equally in both directions.

It keeps its proportions.

Example idea:

- width and height both grow together

This is the most common and safest kind.

Non-uniform scaling

The element scales differently in horizontal and vertical directions.

This can stretch or squash the element.

That can be useful for:

- playful effects
- organic motion
- bounce exaggeration
- custom stylized interactions

But it can also look broken if used carelessly.

Scale and transform origin

This is a very important concept.

When an element scales, **from which point does it scale**?

That is controlled by **transform origin**.

Possible origins:

- center
- top
- bottom
- left
- right
- top left
- bottom center
- etc.

Why this matters

If a button scales from the center, it grows evenly outward.

If a dropdown panel scales from the top, it can feel like it is unfolding downward.

If a line scales from the left, it can feel like it is drawing itself from left to right.

This one concept dramatically changes the feel of an animation.

Scale and realism

Scale can create a sense of depth.

For example:

1. something larger feels closer
2. something smaller feels farther away
3. scaling during scroll can simulate parallax-like depth
4. scaling background and foreground differently can create spatial richness

Even though this is only visual illusion, it feels meaningful to users.

Common mistakes with scale

1. Over-scaling on hover

If a card grows too much on hover, it can feel clumsy and can overlap neighboring content awkwardly.

2. Scaling text too aggressively

Text scaling can become blurry, heavy, or awkward if overused.

3. Ignoring transform origin

An element may appear to grow from the wrong place, making the motion feel unnatural.

4. Using scale without enough duration tuning

If scale happens too fast, it can feel like a glitch instead of intentional animation.

Best-practice mindset for scale

> Use scale for emphasis, focus, depth, and tactile feedback.

Think of it as a way to make interfaces feel more alive—not as a way to constantly enlarge everything.

2.3.3 Rotation

What “rotation” means in animation

Rotation means turning an element around a pivot point.

Examples:

- a plus icon rotates into an X
- an arrow rotates downward when an accordion opens
- a card tilts slightly on hover
- an object spins
- a loader rotates continuously

Rotation is one of the simplest animation types, but it can range from **very subtle** to **very dramatic**.

Visual intuition for rotation

Again, imagine a paper card on a table.

- Position = slide it
- Scale = resize it
- Rotation = turn it around a point

This point is usually its center, but not always.

Rotation and transform origin

Just like with scale, rotation depends heavily on **transform origin**.

If an element rotates around its center, it spins in place.

If it rotates around one edge, it behaves more like a hinged object.

Examples:

- a menu chevron rotates around center
- a door-like panel rotates from the left edge
- a dropdown could appear to swing down from the top edge

This is a huge part of making motion feel natural.

Common uses of rotation

1. Icon state changes

Very common in UI:

- chevron rotates when accordion opens
- plus rotates into close icon
- arrow rotates to show expanded/collapsed state

This is practical because it communicates state clearly.

2. Decorative motion

Background shapes or badges can rotate slightly for energy.

3. Loading indicators

Continuous rotation is often used for spinners.

4. 3D-style interaction

Slight tilt on hover can create a modern interactive feel.

Subtle rotation vs dramatic rotation

Subtle rotation

Small values like:

- `2deg`
- `5deg`
- `10deg`

can make interfaces feel dynamic without being distracting.

Dramatic rotation

Larger values like:

- `90deg`
- `180deg`
- `360deg`

are more noticeable and should be used intentionally.

Rotation communicates meaning

Rotation is not just visual movement—it often implies **state change**.

For example:

- arrow rotates down = content expanded
- arrow rotates up = content collapsed
- plus rotates 45 degrees = close action
- spinning loader = processing

So rotation can function as both:

1. **animation**
2. **communication**

That makes it especially useful in UI design.

Rotation can easily be overused

Beginners sometimes use too much spinning because it is visually obvious and fun.

But on professional sites, excessive rotation often feels:

- distracting
- gimmicky
- unrefined
- difficult to read visually

In most interface work, rotation is best when it is:

- subtle
- meaningful
- supporting a state change

2D vs 3D-feeling rotation

Even before advanced 3D transforms, you should understand that some rotations feel flat while others feel spatial.

Examples:

- flat icon spinning = simple 2D rotation

- card tilting slightly = more depth-oriented feeling
- layered elements rotating differently = richer composition

You do not need advanced math for this at the beginning. Just understand:

> rotation can either feel like “spinning” or like “tilting,” depending on how you use it.

Common mistakes with rotation

1. Rotating text too much

This often hurts readability.

2. Using rotation where simpler motion would work better

Sometimes a small position shift or opacity change is cleaner.

3. Incorrect pivot point

If the transform origin is wrong, the rotation can feel awkward and fake.

4. Too much continuous motion

Constant rotation draws attention forever, which can become annoying.

Best-practice mindset for rotation

>

> Rotation works best when it supports state, direction, or subtle energy.

Use it where turning makes conceptual sense.

2.3.4 Opacity

What “opacity” means in animation

Opacity controls **how visible** an element is.

Typical values:

- `1` = fully visible
- `0` = fully invisible

Values between those create partial transparency.

Opacity is one of the most important animation properties because it is often combined with almost everything else.

Why opacity is so useful

Opacity lets elements:

- appear gently
- disappear smoothly
- feel less abrupt
- blend into motion naturally

Without opacity, movement alone can feel harsh.

For example:

- if text just appears instantly, it can feel mechanical

- if text moves slightly and fades in, it feels more polished

That is why “fade in” is such a common pattern.

Opacity does not equal display

This distinction is very important.

If something has `opacity: 0`, it is visually invisible—but it may still:

- exist in the document
- occupy space
- be clickable
- be focusable
- affect interaction depending on setup

This is different from things like:

- `display: none`
- `visibility: hidden`

So visually hidden is not always functionally removed.

That matters in real projects, especially for accessibility and interaction.

Common uses of opacity

1. Fade in

An element gradually becomes visible.

Used for:

- text reveals
- image reveals
- sections entering viewport
- modals

2. Fade out

An element gradually disappears.

Used for:

- closing overlays
- removing notifications
- transitioning between states

3. Crossfading

One element fades out while another fades in.

This is useful for galleries, tab content, sliders, and testimonial swaps.

4. Softening background elements

Lower opacity can de-emphasize secondary content.

Opacity works best in combination

Opacity alone can be useful, but it is often strongest when combined with other properties.

For example:

1. **opacity + position**
   Fade up reveal
2. **opacity + scale**
   Pop-in effect
3. **opacity + rotation**
   Soft icon transition
4. **opacity + blur**
   Atmospheric reveal, though blur introduces more complexity and performance considerations

---

Why opacity makes animation feel smoother

Opacity helps the brain accept visual change more comfortably.

If an element goes from “not visible” to “visible” gradually, the transition feels more natural than instant switching.

That is especially helpful for:

- page sections
- modal dialogs
- menus
- dynamic content changes

Common mistakes with opacity

1. Fading while leaving interaction active

An invisible element may still be clickable if not handled properly.

2. Making text too transparent

Text at low opacity can become hard to read and may hurt accessibility.

3. Overusing fade-only animation

If everything only fades, the site can feel flat and repetitive.

4. Using long fades everywhere

Slow fades may feel elegant in one place, but sluggish if repeated too often.

Opacity and design quality

Opacity is often associated with “elegant” animation because it softens entry and exit. But elegance comes from **restraint**.

A tiny fade combined with thoughtful timing is often enough.

Best-practice mindset for opacity

> Use opacity to control visual presence and soften transitions.

Think of opacity as the difference between an element *popping in mechanically* and *arriving naturally*.

2.3.5 Timing

What “timing” means in animation

Timing refers to **how the animation unfolds over time**.

This includes:

1. **duration** — how long the animation lasts
2. **easing** — how the speed changes during the animation

Timing is one of the biggest reasons why one animation feels polished and another feels awkward.

You can animate the same properties:

- same position
- same scale
- same opacity

but if the timing is bad, the result will still feel wrong.

Duration

What duration is

Duration is simply the total time an animation takes.

Examples:

- very short
- moderate
- long

In practical web animation, many UI animations are fairly short, but the “right” duration depends on the context.

How duration changes perception

Short duration

Feels:

- fast
- responsive
- sharp
- sometimes abrupt

Good for:

- button feedback
- hover effects
- small icon changes

Medium duration

Feels:

- smooth
- balanced
- intentional

Good for:

- text reveals
- card entrances
- section transitions

Long duration

Feels:

- cinematic
- dramatic
- calm
- sometimes slow or heavy

Good for:

- hero animations
- storytelling sections
- premium brand moments

But too many long animations make a site feel sluggish.

Easing

What easing is

Easing defines **how speed changes during the animation**.

An animation does not have to move at the same speed from start to finish.

That is the key idea.

Easing can make motion feel:

- natural
- mechanical

- energetic
- soft
- heavy
- playful

Why easing matters so much

In the real world, objects rarely:

- start instantly at full speed
- move perfectly evenly
- stop instantly without transition

Natural motion usually includes:

1. acceleration
2. deceleration
3. momentum-like feeling

Easing brings some of that believability into digital interfaces.

Common easing styles conceptually

Linear

Moves at constant speed.

Useful in some cases, such as:

- continuous loops
- marquees
- mechanical effects

But often feels unnatural for UI entry and exit motion.

Ease out

Starts faster and ends slower.

This is extremely common for entrance animations because the element settles nicely into place.

Ease in

Starts slower and speeds up.

Often useful for exits, because it feels like the element is leaving with increasing momentum.

Ease in out

Starts gently, speeds up, then slows down.

Often useful for balanced transitions.

Timing and emotional tone

Timing changes the emotional feel of an animation:

- **fast + sharp** = responsive, technical
- **medium + smooth** = polished, modern
- **slow + soft** = elegant, luxurious
- **springy or bouncy** = playful, energetic

This means timing is not just technical—it is part of branding and UX.

A useful way to think about timing

Ask yourself:

- *Should this feel instant or noticeable?*
- *Should this feel energetic or calm?*
- *Is this an interaction or a presentation?*
- *Should users admire this motion, or barely notice it?*

A button hover should usually feel quick.

A homepage hero reveal can be slower and more expressive.

Common timing mistakes

1. Everything takes the same duration

This makes motion feel robotic.

2. Everything is too slow

This is one of the most common beginner issues.

What feels “smooth” in your imagination can feel “laggy” in real use.

3. Using linear easing for all UI motion

This often feels unnatural.

4. No consistency

If every component has wildly different timing, the site feels uncoordinated.

Best-practice mindset for timing

> Timing is what makes animation feel intentional rather than accidental.

You are not only deciding *what moves*, but *how it behaves in time*.

2.3.6 Delay

What “delay” means in animation

Delay means waiting a certain amount of time **before** an animation begins.

The animation does not start immediately—it pauses first.

This sounds simple, but delay has a major effect on rhythm and clarity.

Why delay is useful

Delay can help you:

1. guide attention
2. create hierarchy
3. prevent too many things from happening at once
4. make motion feel staged and intentional
5. connect related elements in a meaningful order

Example of no delay vs delay

Imagine a hero section with:

- heading
- paragraph
- button
- image

If everything starts at once

The result may feel:

- busy

- noisy
- harder to follow

If elements start with small delays

The result may feel:

- organized
- cinematic
- easier to understand

The user’s eye gets a clearer path through the content.

Delay creates hierarchy

Suppose you animate these elements in order:

1. heading
2. supporting text
3. button

This sequence tells the user:

- first read the main message
- then read the supporting information
- then notice the call to action

That is not just animation. That is **communication design**.

Small delays vs large delays

Small delays

These are often best for interface work.

They create rhythm without making the site feel slow.

Large delays

These can feel dramatic, but they can also frustrate users if they are forced to wait too long.

This is especially risky for:

- navigation
- forms
- buttons
- important content

In UX, delays should usually support clarity—not slow people down unnecessarily.

Delay in hover interactions

A little caution here:

For hover animations, delays can be helpful in some rare cases, but often they make the interface feel less responsive.

For example:

- hover should usually react quickly
- if the user hovers and nothing happens immediately, it may feel broken

So delay is often more useful for:

- entrance animations
- staggered reveals
- orchestrated timelines

than for immediate interactions.

Delay and pacing

Delay contributes to the **pacing** of a whole animated experience.

Think like a film editor:

- what should happen first?

- what should happen second?
- should there be a pause?
- should multiple things overlap?
- should the sequence feel quick or luxurious?

Delay helps answer those questions.

Common mistakes with delay

1. Too much delay

Users should not have to wait for content unnecessarily.

2. Delaying important information

If key text or controls appear too late, the site can feel annoying.

3. Using delay everywhere

This can make the entire site feel sluggish.

4. Delay without purpose

If you cannot explain why the delay exists, it might not need to be there.

Best-practice mindset for delay

> Delay is a rhythm tool, not a decoration.

Use it to improve order, hierarchy, and attention flow.

2.3.7 Sequencing

What “sequencing” means in animation

Sequencing means deciding **the order and relationship of multiple animations**.

This is where animation starts to feel like a real system rather than isolated effects.

Instead of asking:

- “How does this one element animate?”

you ask:

- “How do these elements animate together?”

That is sequencing.

Why sequencing matters

Most website animations do not involve just one element.

A typical section may contain:

- heading
- text
- image
- button
- background decoration
- cards
- icons

If all of these animate independently without a plan, the result can feel chaotic.

Sequencing gives structure.

Three basic sequencing relationships

1. Simultaneous

Everything starts at the same time.

This can work when:

- the motion is simple
- the group should feel unified
- there are only a few elements

But with many elements, this can become visually noisy.

2. Sequential

One animation starts after another.

This creates:

- order
- hierarchy
- storytelling
- guided attention

3. Overlapping

A second animation begins before the first one fully finishes.

This is often the sweet spot for polished motion.

It feels:

- fluid
- connected
- professional

Too much strict waiting can feel stiff.
Too much simultaneity can feel messy.
Overlap often gives the best balance.

Sequencing creates storytelling

Even simple website sections can tell a mini-story through sequencing.

For example:

1. background image becomes visible
2. headline moves in
3. paragraph fades up
4. button appears
5. decorative line expands

This gives the user a visual reading order.

Sequencing and hierarchy

The order of motion communicates importance.

Usually:

- primary content animates first
- secondary content animates after
- decorative content either animates subtly or later

If a decorative flourish animates before the main headline, you may accidentally draw attention to the wrong thing.

Sequencing and rhythm

Sequencing is not only about order, but also about spacing in time.

Questions to consider:

- Should each step wait fully for the previous one?
- Should they overlap?
- Should the sequence feel tight and snappy?
- Should it breathe and feel luxurious?

This is rhythm.

Sequencing patterns you'll use often

Pattern 1: Heading → text → button

Very common for hero sections and banners.

Pattern 2: Staggered card reveal

Cards enter one after another.

This helps users scan a set naturally.

Pattern 3: Image first, text second

Useful when the visual should establish context.

Pattern 4: Text first, image second

Useful when the message is more important than decoration.

Pattern 5: Parent and child sequencing

A section becomes visible, then internal elements animate in.

This creates structure and avoids clutter.

Sequencing and user attention

Animation should support the natural reading flow.

For left-to-right languages, users often scan:

1. top to bottom
2. left to right

So sequencing that follows that logic can feel intuitive.

But you can also intentionally break that order if a design calls for it.

The key is that the sequence should feel **motivated**, not random.

Common mistakes with sequencing

1. Too many animated elements

If every little thing has its own sequence, users become overwhelmed.

2. No hierarchy

If everything is treated equally, nothing feels important.

3. Overly long sequences

Users may have to wait too long before the interface feels ready.

4. Decorative elements dominating the sequence

The main content should usually win.

5. No consistency across the site

If each section has a completely different sequencing logic, the site can feel disjointed.

Best-practice mindset for sequencing

> Sequencing is choreography.

You are arranging motion so the user’s eye moves through content in a clear, intentional way.

How all seven concepts work together

These concepts rarely appear alone. Most useful animations combine several at once.

For example, a common “fade up” reveal includes:

1. **position** — starts slightly lower
2. **opacity** — starts invisible
3. **timing** — moderate duration with smooth easing
4. **delay** — maybe starts after the previous element
5. **sequencing** — headline first, then paragraph, then button

Another example, a hover card might use:

1. **scale** — card grows slightly
2. **position** — image shifts upward
3. **rotation** — icon tilts a little
4. **timing** — quick and responsive

5. **sequencing** — image and text move together, icon follows slightly

So in practice, animation design is often about building combinations.

A simple mental framework for analyzing any animation

Whenever you see an animation on a website, ask these seven questions:

1. **Position**
   Is it moving somewhere?
2. **Scale**
   Is it getting bigger or smaller?
3. **Rotation**
   Is it turning?
4. **Opacity**
   Is it fading in or out?
5. **Timing**
   How long does it take, and how does the speed feel?
6. **Delay**
   Does it wait before starting?
7. **Sequencing**
   How does it relate to other animations nearby?

If you train yourself to analyze motion this way, you will understand GSAP much faster later.

A practical website-oriented example

Imagine a section in BricksBuilder containing:

- heading
- text
- button

- image

A polished entrance might be described like this:

1. **Heading**
   1. starts `20px` lower
   2. starts at `opacity: 0`
   3. moves into place and fades in
2. **Text**
   1. starts slightly after the heading
   2. also moves upward a bit
   3. fades in with similar timing
3. **Button**
   1. appears after text
   2. maybe scales from slightly smaller
   3. fades in quickly
4. **Image**
   1. starts slightly larger
   2. fades in
   3. settles to normal scale

That sounds “complex,” but really it is just a combination of the seven concepts.

What this means for your future GSAP learning

When you later learn GSAP, you will see code that controls:

- `x`, `y`
- `scale`
- `rotation`
- `opacity`
- `duration`
- `delay`
- timeline order

And instead of seeing mysterious technical words, you’ll understand the visual logic behind them.

That is exactly why this section comes **before** GSAP syntax.

Summary

The core visual concepts of animation are:

1. **Position**
   Changes where an element is
2. **Scale**
   Changes how large or small it appears
3. **Rotation**
   Changes its angle or direction
4. **Opacity**
   Changes how visible it is
5. **Timing**
   Controls duration and speed behavior
6. **Delay**
   Controls when the animation begins
7. **Sequencing**
   Controls how multiple animations are arranged together

If you truly understand these seven ideas, you already understand the **language of animation** □□

GSAP will then become the tool you use to express that language precisely.

Recommended mindset before moving on

Before learning actual GSAP code, try to practice this:

1. Visit a few modern websites.
2. Observe one animation at a time.
3. Describe it using the seven concepts.
4. Ask yourself why the designer made those choices.
5. Think about whether the animation helps clarity, hierarchy, or emotion.

That habit will make you much stronger—not just at using GSAP, but at designing motion well.

Revision #1
Created 2026-04-25 19:55:16 UTC by art10m
Updated 2026-04-25 19:59:24 UTC by art10m