Functions

Why This Matters

Functions are the beating heart of mathematics and programming.

Everything you've learned so far—numbers, algebra, coordinates—comes together in functions. A function is:

- **A relationship** between inputs and outputs
- **A rule** that transforms data
- **A mapping** from one set to another

As a developer, you already use functions constantly:

```
function double(x) {
  return x * 2;
}
```

Mathematical functions work **exactly the same way**. Understanding them deeply unlocks calculus, data science, ML, and advanced programming.

The Big Picture: Input → Transform → Output

The Machine Metaphor

Think of a function as a **machine**:

```
    INPUT
      ↓
   ┌──────────┐
   │ FUNCTION │   (transformation rule)
   └──────────┘
      ↓
    OUTPUT
```

Example: "Doubling machine"

```
Input: 3
  ↓
[×2]
  ↓
Output: 6
```

Programming Analogy

```
// Function (math)
f(x) = 2x + 1

// Function (code)
function f(x) {
```

```
  return 2*x + 1;
}

// Both do the same thing:
f(3) → 2(3) + 1 → 7
```

Math functions and code functions are the same concept.

---

1. Function Notation

Standard Form

```
f(x) = ...

f     = function name
x     = input variable (argument)
f(x)  = output (return value)
```

Read as: "f of x" or "f at x"

Examples

```
f(x) = 2x + 1

f(3) = 2(3) + 1 = 7
f(0) = 2(0) + 1 = 1
f(-2) = 2(-2) + 1 = -3
```

Multiple Function Names

```
f(x) = x²
g(x) = 3x - 2
h(x) = √x
```

Different names for different functions (like variable names in code).

Other Notation

Sometimes you'll see:

```
y = f(x)
y = 2x + 1
```

Here, `y` is the output when input is `x` .

---

2. What Makes Something a Function?

The Key Rule: One Input → One Output

A function must give exactly ONE output for each input.

Valid function:

```
f(x) = x²

f(2) = 4    (always 4, never anything else)
f(2) = 4    (consistent)
```

NOT a function:

```
"What is ±√x?"

√4 = 2 or -2  (two outputs for one input)
```

This is a **relation**, not a function.

Visual Test: Vertical Line Test

If a vertical line intersects a graph more than once, it's NOT a function.

```
Function (passes test):     Not a function (fails test):
    y                            y
    |                            |
    |     •                      |   •
    |   •                        |  • •  ← Two y-values at same x
    | •                          | •   •
    |•                           |•     •
────┴──────────── x          ────┴──────────── x
```

Why This Matters

Functions are predictable: Same input always gives same output.

```
// Good function (predictable)
function square(x) {
  return x * x;
}

square(5);  // Always 25

// Bad "function" (unpredictable)
function random(x) {
  return Math.random();  // Different each time!
}
```

Math functions are **pure functions** in programming terms.

---

3. Domain and Range

Domain: All Valid Inputs

The **domain** is the set of all input values that work.

Examples:

```
f(x) = x + 5
Domain: All real numbers (ℝ)
(You can add 5 to any number)

g(x) = √x
Domain: x ≥ 0
(Can't take square root of negative in real numbers)

h(x) = 1/x
Domain: x ≠ 0
(Can't divide by zero)
```

Range: All Possible Outputs

The **range** is the set of all output values the function can produce.

Examples:

```
f(x) = x²
Domain: All real numbers
Range: y ≥ 0 (squares are never negative)

g(x) = sin(x)
Domain: All real numbers
Range: -1 ≤ y ≤ 1 (sine oscillates between -1 and 1)
```

Visual Understanding

```
Domain (inputs):     Function:      Range (outputs):
    x-axis             graph            y-axis

───────────────    f(x) = x²      ───────────────
<— domain: ℝ —>                   ↑ range: y≥0 ↑
```

Programming Analogy

```
// Domain: integers (implicitly)
// Range: booleans
function isEven(n) {
  return n % 2 === 0;
}

// Domain: numbers (type system)
// Range: numbers
function double(x) {
  return x * 2;
}
```

Type signatures in TypeScript are like domain/range:

```
function f(x: number): number {
  return x * 2;
}
// Domain: number, Range: number
```

---

4. Evaluating Functions

Substitution

Replace the variable with the input value:

```
f(x) = 3x² - 2x + 1

Find f(4):
f(4) = 3(4)² - 2(4) + 1
     = 3(16) - 8 + 1
     = 48 - 8 + 1
     = 41
```

Multiple Inputs

```
f(x) = x² - 5

f(0) = 0² - 5 = -5
f(1) = 1² - 5 = -4
f(2) = 4 - 5 = -1
f(3) = 9 - 5 = 4
```

Variable Expressions

You can input expressions, not just numbers:

```
f(x) = x² + 1

f(a) = a² + 1
f(2x) = (2x)² + 1 = 4x² + 1
f(x+h) = (x+h)² + 1 = x² + 2xh + h² + 1
```

This is crucial for calculus.

---

5. Graphing Functions

Every Function Has a Graph

The graph is **all points (x, f(x))**:

```
f(x) = x + 2

Points:
x:  -2  -1   0   1   2
```

```
y:   0   1   2   3   4

Graph:
    y
    │
  4 ├─────●
  3 ├───●
  2 ├─●
  1 ├●
  0 ├●
────┼─────────▶ x
   -2 -1 0 1 2
```

Common Function Shapes

Linear: f(x) = mx + b

```
    │
    │   ╱
    │  ╱  (straight line)
    │ ╱
────┼╱──────
    │
```

Quadratic: f(x) = x²

```
    │
    │   ●
    │  ╱ ╲
    │ ╱   ╲
────┼╱─────╲────
    │       (parabola)
```

Cubic: f(x) = x³

```
    │     ╱
    │    ╱
    │   ╱  (S-curve)
────┼──╱────
    │ ╱
    │╱
```

Exponential: f(x) = 2ˣ

```
    │       ╱
    │      ╱
    │    ╱╱  (rapid growth)
────┼───╱────
    │__╱
```

Logarithmic: f(x) = log(x)

```
     │
     │   ╱── (slow growth)
     │  ╱
─────┼─╱──────
     │╱
```

Programming: Plotting Functions

```
// Generate points
function plotFunction(f, xMin, xMax, step) {
  const points = [];
  for (let x = xMin; x <= xMax; x += step) {
    points.push({ x, y: f(x) });
  }
  return points;
}

// Example
const f = x => x**2;
const points = plotFunction(f, -5, 5, 0.5);
// Returns [{x:-5, y:25}, {x:-4.5, y:20.25}, ...]
```

---

6. Composition of Functions

What It Means

Apply one function, then apply another to the result.

Notation: (f ∘ g)(x) = f(g(x))

Read as: "f composed with g" or "f of g of x"

Visual

```
    x
    ↓
   [g]  ← First function
    ↓
   g(x)
    ↓
   [f]  ← Second function
    ↓
  f(g(x))
```

Example

```
f(x) = 2x + 1
g(x) = x²
```

```
Find (f ∘ g)(3):

Step 1: Apply g first
g(3) = 3² = 9

Step 2: Apply f to result
f(9) = 2(9) + 1 = 19

So: (f ∘ g)(3) = 19
```

General Form

```
(f ∘ g)(x) = f(g(x))

Replace x in f with the entire g(x):

f(x) = 2x + 1
g(x) = x²

f(g(x)) = 2(x²) + 1 = 2x² + 1
```

Order Matters!

```
f(g(x)) ≠ g(f(x))

f(x) = 2x
g(x) = x + 1

f(g(x)) = 2(x+1) = 2x + 2
g(f(x)) = 2x + 1

Different results!
```

Programming Analogy

```
const f = x => 2*x + 1;
const g = x => x**2;

// Composition: f(g(x))
const compose = (f, g) => x => f(g(x));

const fog = compose(f, g);
fog(3);  // f(g(3)) = f(9) = 19

// Also chainable:
const result = f(g(3));
```

---

7. Inverse Functions

What They Are

An **inverse function** undoes what the original function does.

Notation: $f^{-1}(x)$ (read as "f inverse of x")

```
f(x) = 2x      (doubles)
f⁻¹(x) = x/2   (halves, undoing the double)

f(5) = 10
f⁻¹(10) = 5  (undoes it)
```

The Key Property

```
f(f⁻¹(x)) = x
f⁻¹(f(x)) = x

They cancel each other out
```

Visual

```
    x  ──[f]──→  f(x)
       ←─[f⁻¹]─

Going forward then backward returns to x
```

Examples

```
f(x) = x + 5       f⁻¹(x) = x - 5  (subtract undoes add)
f(x) = 3x          f⁻¹(x) = x/3    (divide undoes multiply)
f(x) = x²          f⁻¹(x) = √x     (root undoes square)
f(x) = eˣ          f⁻¹(x) = ln(x)  (log undoes exponential)
```

Finding Inverse Functions

Method:

1. Replace f(x) with y
2. Swap x and y
3. Solve for y
4. Replace y with $f^{-1}(x)$

Example: Find inverse of f(x) = 2x + 3

```
Step 1: y = 2x + 3
Step 2: x = 2y + 3  (swap x and y)
Step 3: Solve for y
   x = 2y + 3
   x - 3 = 2y
```

```
   y = (x - 3) / 2

Step 4: f⁻¹(x) = (x - 3) / 2
```

Verify:

```
f(5) = 2(5) + 3 = 13
f⁻¹(13) = (13 - 3) / 2 = 5 ✓
```

Not All Functions Have Inverses

A function must be **one-to-one** (each output comes from exactly one input).

Bad example: $f(x) = x^2$

```
f(2) = 4
f(-2) = 4   ← Same output from two inputs

If you try to invert:
f⁻¹(4) = 2 or -2?  (ambiguous!)
```

Solution: Restrict domain to $x \geq 0$, then $f^{-1}(x) = \sqrt{x}$ works.

Programming Analogy

```
// Function and its inverse
function encode(x) {
  return x * 2 + 3;
}

function decode(y) {
  return (y - 3) / 2;
}

const original = 10;
const encoded = encode(original);   // 23
const decoded = decode(encoded);    // 10
console.log(decoded === original);  // true

// Encryption/decryption is inverse functions
```

---

8. Types of Functions

Linear Functions

```
f(x) = mx + b

m = slope
b = y-intercept
```

```
Graph: Straight line
```

Quadratic Functions

```
f(x) = ax² + bx + c

Graph: Parabola (U-shaped)
```

Polynomial Functions

```
f(x) = aₙxⁿ + ... + a₁x + a₀

Sum of power terms
```

Rational Functions

```
f(x) = P(x) / Q(x)

Ratio of polynomials
Example: f(x) = (x+1)/(x-2)
```

Exponential Functions

```
f(x) = aˣ

Base raised to variable power
Example: f(x) = 2ˣ
```

Logarithmic Functions

```
f(x) = log_b(x)

Inverse of exponential
```

Trigonometric Functions

```
f(x) = sin(x), cos(x), tan(x), ...

Based on angles and circles
```

Piecewise Functions

Different rules for different inputs:

```
        ⎧ x + 1,  if x < 0
f(x) =  ⎨ x²,     if 0 ≤ x < 2
        ⎩ 5,      if x ≥ 2
```

Programming:

```
function f(x) {
  if (x < 0) return x + 1;
  if (x < 2) return x**2;
  return 5;
}
```

---

9. Function Transformations

Vertical Shift

```
f(x) + k  → Shift up by k
f(x) - k  → Shift down by k

If f(x) = x²:
f(x) + 2 = x² + 2  (parabola shifted up 2)
```

Horizontal Shift

```
f(x - h) → Shift right by h
f(x + h) → Shift left by h

If f(x) = x²:
f(x - 2) = (x-2)²  (parabola shifted right 2)
```

Note: Sign is opposite! +2 shifts left, -2 shifts right.

Vertical Stretch/Compression

```
a·f(x) where a > 0

a > 1: Stretch (taller)
0 < a < 1: Compress (shorter)
a < 0: Stretch + flip

If f(x) = x²:
2f(x) = 2x²     (twice as tall)
0.5f(x) = 0.5x² (half as tall)
```

Horizontal Stretch/Compression

```
f(bx) where b > 0

b > 1: Compress (narrower)
0 < b < 1: Stretch (wider)

If f(x) = x²:
```

```
f(2x) = (2x)² = 4x²  (narrower)
f(0.5x) = (0.5x)²    (wider)
```

Reflection

```
-f(x)  → Flip over x-axis
f(-x)  → Flip over y-axis

If f(x) = x²:
-f(x) = -x²  (upside-down parabola)
f(-x) = (-x)² = x²  (same, since x² is even)
```

---

10. Real-World Applications

Physics: Position Function

```
s(t) = -16t² + v₀t + s₀

s(t) = position at time t
v₀ = initial velocity
s₀ = initial position

Models projectile motion
```

Economics: Cost Function

```
C(x) = fixed_cost + variable_cost × x

C(100) = total cost to produce 100 units
```

Computer Science: Hash Functions

```
hash(x) → fixed-size output

Used in: hash tables, cryptography, checksums
```

Data Processing: Map Function

```javascript
// Array.map is function application
const data = [1, 2, 3, 4];
const doubled = data.map(x => x * 2);  // [2, 4, 6, 8]

// This is: f(x) = 2x applied to each element
```

Machine Learning: Activation Functions

```
σ(x) = 1 / (1 + e⁻ˣ)  (sigmoid function)

Maps any input to (0, 1)
Used in neural networks
```

Game Development: Easing Functions

```
// Linear
f(t) = t

// Ease-in (quadratic)
f(t) = t²

// Ease-out
f(t) = 1 - (1-t)²

t ∈ [0, 1] (time)
```

Common Mistakes & Misconceptions

❌ "f(x) and f·x are the same"

No!

```
f(x) = function applied to x
f·x = f multiplied by x (different notation)
```

❌ "f(a + b) = f(a) + f(b)"

Usually **false**:

```
f(x) = x²

f(2 + 3) = f(5) = 25
f(2) + f(3) = 4 + 9 = 13

25 ≠ 13
```

❌ "(f ∘ g)(x) = f(x) · g(x)"

No! Composition ≠ multiplication:

```
(f ∘ g)(x) = f(g(x))  (composition)
(f · g)(x) = f(x) · g(x)  (multiplication)
```

❌ "All functions have inverses"

Only **one-to-one** functions have inverses.

❌ "Domain is always all real numbers"

Many functions have restrictions:

```
√x: x ≥ 0
1/x: x ≠ 0
log(x): x > 0
```

Tiny Practice

Evaluate:

1. $f(x) = 3x - 2$, find $f(5)$
2. $g(x) = x^2 + 1$, find $g(-3)$
3. $h(x) = 2x^2 - x + 4$, find $h(2)$

Composition:

4. $f(x) = 2x$, $g(x) = x + 1$, find $f(g(3))$
5. $f(x) = x^2$, $g(x) = 3x$, find $(f \circ g)(2)$

Inverse:

6. Find the inverse of $f(x) = 3x - 6$
7. If $f(x) = x + 5$, what is $f^{-1}(10)$?

Domain/Range:

8. What is the domain of $f(x) = \sqrt{(x - 3)}$?
9. What is the range of $f(x) = x^2 + 1$?

▶ Answers

Summary Cheat Sheet

Function Basics

```
f(x) = ...    (function notation)

Input: x
Output: f(x)
Graph: all points (x, f(x))
```

Domain and Range

```
Domain: Valid inputs (x-values)
Range: Possible outputs (y-values)
```

Operations

Operation	Notation	Meaning
Evaluate	$f(3)$	Substitute 3 for x
Compose	$(f \circ g)(x)$	$f(g(x))$

Inverse	$f^{-1}(x)$	Undoes f
Add	(f + g)(x)	f(x) + g(x)
Multiply	(f · g)(x)	f(x) · g(x)

Transformations

```
f(x) + k    → Shift up
f(x - h)    → Shift right
a·f(x)      → Vertical stretch
f(bx)       → Horizontal compression
-f(x)       → Flip over x-axis
f(-x)       → Flip over y-axis
```

Programming Parallels

```
// Math function
f(x) = 2x + 1

// Code function
const f = x => 2*x + 1;

// Composition
const compose = (f, g) => x => f(g(x));

// Inverse (conceptually)
encode(x) and decode(y) are inverses
```

Next Steps

Functions are the foundation of everything in advanced mathematics. You now understand:

- What functions are
- How to evaluate and graph them
- Composition and inverses
- Transformations

Next, we'll apply this to **Linear Functions**—the simplest and most fundamental type.

Continue to: 07-linear-functions.md