Probability Basics (Developer Edition)

What Problem This Solves

Probability helps you reason about uncertainty in systems.

Every time you deal with:

- **Failures**: Will this request succeed? Should I retry?
- **Performance**: What's the cache hit rate?
- **Testing**: How likely is this bug to appear?
- **Capacity**: What load can we handle?
- **Security**: How strong is this password?

...you're reasoning about probability.

Probability turns "I don't know" into "Here's what I can expect."

Intuition & Mental Model

Think: Frequency Over Many Trials

Probability ≈ "What fraction of the time does this happen?"

```
Coin flip: P(heads) = 0.5
→ In 1000 flips, expect ~500 heads

API success: P(success) = 0.99
→ In 1000 requests, expect ~990 successes
```

Not fortune telling—long-run behavior.

Core Concepts

1. Basic Probability

Probability of event A:

```
           # of outcomes where A happens
P(A) = ─────────────────────────────────────
         # of total possible outcomes

Range: 0 ≤ P(A) ≤ 1
```

Example: Rolling a die

```
P(rolling 3) = 1/6 ≈ 0.167
P(rolling even) = 3/6 = 0.5
P(rolling 7) = 0/6 = 0 (impossible)
P(rolling ≤6) = 6/6 = 1 (certain)
```

In code:

```javascript
// Fair coin flip
function flipCoin() {
  return Math.random() < 0.5 ? 'heads' : 'tails';
}

// P(heads) = 0.5
let heads = 0;
for (let i = 0; i < 10000; i++) {
  if (flipCoin() === 'heads') heads++;
}
console.log(heads / 10000); // ~0.5
```

---

2. Complementary Events

P(not A) = 1 - P(A)

```
P(heads) = 0.5
P(tails) = 1 - 0.5 = 0.5

P(request succeeds) = 0.99
P(request fails) = 1 - 0.99 = 0.01
```

Useful for "at least one" problems:

```javascript
// P(at least one success) = 1 - P(all failures)

// P(no failures in 3 tries) given P(fail) = 0.1 per try
const pAllFail = 0.1 * 0.1 * 0.1; // 0.001
const pAtLeastOneSuccess = 1 - pAllFail; // 0.999
```

---

3. Independent Events

Two events are independent if one doesn't affect the other.

```
P(A and B) = P(A) × P(B)   // Only if independent!
```

Example: Multiple coin flips

```
P(heads, then heads) = 0.5 × 0.5 = 0.25
P(three heads in a row) = 0.5 × 0.5 × 0.5 = 0.125
```

In systems:

```javascript
// Independent failures
const dbUptime = 0.99;    // 99% uptime
const apiUptime = 0.98;   // 98% uptime

// Both up (independent)
const bothUp = dbUptime * apiUptime; // 0.9702 (97.02%)
```

```
// At least one down
const atLeastOneDown = 1 - bothUp; // 0.0298 (2.98%)
```

---

4. Conditional Probability

P(A | B) = "Probability of A given B happened"

```
          P(A and B)
P(A|B) = ────────────
            P(B)
```

Example: Login attempts

```
P(hacker | failed login) ≠ P(failed login)

Failed logins happen often (typos)
But given 10 failed logins in a row, P(hacker) increases
```

In code:

```
// Cache effectiveness
const totalRequests = 10000;
const cacheHits = 8000;
const cacheMisses = 2000;
const slowResponses = 300; // All from cache misses

// P(slow response)
const pSlow = slowResponses / totalRequests; // 0.03

// P(slow | cache miss)
const pSlowGivenMiss = slowResponses / cacheMisses; // 0.15

// Conditional probability is higher!
```

---

5. Bayes' Theorem (Intuition)

Flipping conditional probabilities:

```
          P(B|A) × P(A)
P(A|B) = ───────────────
              P(B)
```

Example: False positives in testing

```
Test accuracy: P(positive | bug present) = 0.95
Bug prevalence: P(bug present) = 0.01

If test is positive, P(bug actually present)?

P(bug | positive) = P(positive | bug) × P(bug)
                    ────────────────────────────
```

```
                           P(positive)

Counterintuitive: Even with 95% accurate test,
P(bug | positive) might be low if bugs are rare!
```

Real scenario:

```
// Health monitoring system
const pHighLoadGivenIssue = 0.9;  // Issue → high load
const pIssue = 0.05;               // 5% of time there's an issue
const pHighLoad = 0.2;             // 20% of time load is high

// P(issue | high load)?
const pIssueGivenHighLoad =
  (pHighLoadGivenIssue * pIssue) / pHighLoad;
// = (0.9 × 0.05) / 0.2 = 0.225 (22.5%)

// High load doesn't always mean issue!
```

---

6. Expected Value

Average outcome over many trials:

```
E[X] = Σ (outcome × probability)
```

Example: Dice roll

```
E[dice] = 1×(1/6) + 2×(1/6) + 3×(1/6) + 4×(1/6) + 5×(1/6) + 6×(1/6)
        = (1+2+3+4+5+6) / 6
        = 3.5
```

In systems:

```
// Expected response time
const scenarios = [
  { time: 10, probability: 0.7 },  // Cache hit
  { time: 100, probability: 0.25 }, // Database query
  { time: 1000, probability: 0.05 } // Timeout
];

const expectedTime = scenarios.reduce(
  (sum, {time, probability}) => sum + time * probability,
  0
);
// = 10×0.7 + 100×0.25 + 1000×0.05 = 82ms

// Expected latency: 82ms
```

Software Engineering Connections

1. Retry Logic

```
async function fetchWithRetry(url, maxRetries = 3) {
  const pSuccess = 0.9; // 90% success rate per try

  for (let i = 0; i < maxRetries; i++) {
    try {
      return await fetch(url);
    } catch (error) {
      if (i === maxRetries - 1) throw error;
      // Wait before retry (exponential backoff)
      await sleep(2 ** i * 1000);
    }
  }
}

// P(all 3 attempts fail) = 0.1³ = 0.001
// P(at least one succeeds) = 1 - 0.001 = 0.999 (99.9%)
```

---

2. Load Balancing

```
// Random load balancing
const servers = ['server1', 'server2', 'server3'];

function randomServer() {
  return servers[Math.floor(Math.random() * servers.length)];
}

// Each server gets ~1/3 of traffic (uniform distribution)
// Over 10,000 requests, each expects ~3,333 requests
```

---

3. A/B Testing

```
// Show variant A to 50% of users
function getVariant(userId) {
  const hash = simpleHash(userId);
  return hash % 2 === 0 ? 'A' : 'B';
}

// Track conversions
const results = {
  A: { shown: 5000, converted: 250 }, // 5% conversion
  B: { shown: 5000, converted: 300 }  // 6% conversion
};

// Is B better? Need statistical significance test
// (covered in inferential statistics)
```

---

4. Cache Hit Probability

```
// LRU cache with capacity 1000
const cache = new LRUCache(1000);

// After warmup, track hit rate
let hits = 0, misses = 0;

function getData(key) {
  if (cache.has(key)) {
    hits++;
    return cache.get(key);
  }

  misses++;
  const data = fetchFromDB(key);
  cache.set(key, data);
  return data;
}

// P(cache hit) = hits / (hits + misses)
// Goal: Keep P(hit) > 0.9 (90%)
```

---

5. Password Strength

```
function passwordStrength(password) {
  const charset = {
    lowercase: 26,
    uppercase: 26,
    digits: 10,
    symbols: 32
  };

  let charsetSize = 0;
  if (/[a-z]/.test(password)) charsetSize += charset.lowercase;
  if (/[A-Z]/.test(password)) charsetSize += charset.uppercase;
  if (/[0-9]/.test(password)) charsetSize += charset.digits;
  if (/[^a-zA-Z0-9]/.test(password)) charsetSize += charset.symbols;

  // Possible combinations
  const combinations = charsetSize ** password.length;

  // At 1 billion tries/sec, expected time to crack
  const secondsToCrack = combinations / 1e9;

  return { combinations, secondsToCrack };
}
```

```
// 8-char, all lowercase: 26^8 = 208 billion (~3.5 minutes)
// 8-char, mixed: 94^8 = 6 quadrillion (~197,000 years)
```

---

6. Bloom Filters (Probabilistic Data Structure)

```
class BloomFilter {
  constructor(size) {
    this.bits = new Array(size).fill(false);
    this.size = size;
  }

  add(item) {
    const hash1 = this.hash(item, 0) % this.size;
    const hash2 = this.hash(item, 1) % this.size;
    this.bits[hash1] = true;
    this.bits[hash2] = true;
  }

  mightContain(item) {
    const hash1 = this.hash(item, 0) % this.size;
    const hash2 = this.hash(item, 1) % this.size;
    return this.bits[hash1] && this.bits[hash2];
  }

  hash(item, seed) {
    // Simplified hash function
    let hash = seed;
    for (let char of item) {
      hash = (hash * 31 + char.charCodeAt(0)) % this.size;
    }
    return hash;
  }
}

// False positive rate ≈ (1 - e^(-k*n/m))^k
// k = hash functions, n = elements, m = bit array size

// Trade space for accuracy
```

Common Misconceptions

❌ "Random means unpredictable for individual events"

For single events, yes. But over many trials, very predictable.

```
// Can't predict one flip
flipCoin(); // 'heads' or 'tails'?

// Can predict distribution
```

```javascript
let heads = 0;
for (let i = 0; i < 100000; i++) {
  if (flipCoin() === 'heads') heads++;
}
console.log(heads / 100000); // ~0.5 (very close)
```

❌ "Past events affect independent future events"

Gambler's fallacy: Thinking streak affects probability

```
// "5 heads in a row, tails is 'due'"
// WRONG! Next flip still 50/50

// Coin has no memory
```

❌ "Conditional probability is symmetric"

P(A|B) ≠ P(B|A)

```
P(rain | clouds) ≠ P(clouds | rain)

P(slow response | cache miss) ≠ P(cache miss | slow response)
```

❌ "Low probability means impossible"

Low probability events happen with enough trials:

```
// P(hash collision) = very small
// But with billions of operations, collisions happen

// Birthday paradox: 23 people → 50% chance of shared birthday
// Seems low, but probability adds up
```

❌ "Expected value is the most common outcome"

Expected value might not even be possible:

```
E[dice roll] = 3.5
But you can't roll 3.5!
```

Practical Mini-Exercises

Exercise 1: Uptime Calculation

Your system has three services:

- API: 99.9% uptime (0.999)
- Database: 99.5% uptime (0.995)
- Cache: 99.9% uptime (0.999)

If all three must be up for system to work, what's overall uptime?

▶ Solution

Exercise 2: Retry Strategy

API call has 80% success rate. How many retries to get 99% overall success?

▶ Solution

Exercise 3: Cache Sizing

Your cache hit rate is 85% with size 1000. Each cache hit saves 50ms.

Expected response time if base (no cache) is 60ms?

▶ Solution

Summary Cheat Sheet

Basic Rules

```
// Probability range
0 ≤ P(A) ≤ 1

// Complement
P(not A) = 1 - P(A)

// Independent events
P(A and B) = P(A) × P(B)

// Conditional probability
P(A | B) = P(A and B) / P(B)

// Expected value
E[X] = Σ (outcome × probability)
```

Common Patterns

```
// At least one success in n tries
pSuccess = 1 - (pFail ** n)

// All succeed
pAllSuccess = pSuccess ** n

// Expected time with cache
E[time] = pHit × hitTime + pMiss × missTime

// Uptime calculation
systemUptime = service1Uptime × service2Uptime × ...
```

Quick Reference

Scenario	Formula	Example

Coin flip	P = 0.5	Heads/tails
Dice roll	P = 1/6 per face	Rolling 3
Independent AND	P(A) × P(B)	Both succeed
Independent OR	1 - P(both fail)	At least one succeeds
Retry success	1 - (pFail^n)	3 retries

Next Steps

Probability helps you reason about uncertainty and randomness in systems. You now understand how to calculate likelihoods, expected values, and make informed decisions under uncertainty.

Next, we'll explore **descriptive statistics**—understanding and summarizing data you collect from your systems.

Continue to: 07-descriptive-statistics.md