@fin.cx/calculation

a unified calculation module to always get to the exact same result.

- readme.md for @fin.cx/calculation
- changelog.md for @fin.cx/calculation

readme.md for @fin.cx/calculation

> Professional financial calculations with decimal precision for JavaScript and TypeScript

npm version TypeScript License: MIT

□□ Why @fin.cx/calculation?

JavaScript's floating-point arithmetic can lead to precision errors in financial calculations:

```
// JavaScript floating point problem
0.1 + 0.2 // 0.30000000000000004 □

// With @fin.cx/calculation
calculator.add(0.1, 0.2) // 0.3 □
```

This package provides **accurate financial calculations** using decimal arithmetic, making it perfect for:

- □□ Banking applications
- □□ Investment analysis tools
- □□ Financial reporting systems
- □□ Accounting software
- □□ Payment processing systems
- □□ Trading platforms

□ Features

- □□ **Decimal Precision**: Built on decimal.js for accurate calculations

- □□ **Comprehensive Functions**: Time value of money, interest calculations, loan amortization, and more
- □□ **Currency Support**: Built-in currency formatting and conversion
- □□ **Cross-Platform**: Works in Node.js and browsers
- □□ **TypeScript First**: Full type safety and IntelliSense support
- □□ **Battle-Tested**: Extensive test suite ensures reliability
- □□ **Zero Dependencies**: Only depends on decimal.js for core functionality

□□ Installation

```
npm install @fin.cx/calculation
# or
yarn add @fin.cx/calculation
# or
pnpm add @fin.cx/calculation
```

□□ Quick Start

```
import { Calculator, Financial, Currency } from '@fin.cx/calculation';

// Basic calculations with precision
const calc = new Calculator();
const result = calc.add(0.1, 0.2); // 0.3 (exact)

// Financial calculations
const financial = new Financial();
const payment = financial.payment(200000, 0.045/12, 360); // Monthly mortgage payment

// Currency operations
const currency = new Currency();
currency.setExchangeRate('USD', 'EUR', 0.85);
const euros = currency.convert(100, 'USD', 'EUR'); // Convert $100 to €85
```

□□ Core Classes

Calculator - Precision Arithmetic

```
const calc = new Calculator({ precision: 10 });

// Basic operations
calc.add(10.5, 20.3);           // 30.8
calc.subtract(100, 45.5);       // 54.5
calc.multiply(15, 3.5);         // 52.5
calc.divide(100, 3);            // 33.3333333333...

// Advanced operations
calc.power(2, 8);               // 256
calc.sqrt(16);                  // 4
calc.ln(Math.E);                // 1
calc.round(3.14159, 2);         // 3.14
```

Financial - Time Value of Money

```
const financial = new Financial();

// Present Value - How much is $10,000 in 5 years worth today at 5% interest?
const pv = financial.presentValue(10000, 0.05, 5); // $7,835.26

// Future Value - How much will $5,000 be worth in 10 years at 7% interest?
const fv = financial.futureValue(5000, 0.07, 10); // $9,835.76

// Loan Payment - Monthly payment for $200,000 mortgage at 4.5% for 30 years
const pmt = financial.payment(200000, 0.045/12, 360); // $1,013.37

// Net Present Value - Is this investment worth it?
const cashFlows = [-50000, 15000, 15000, 15000, 15000, 20000];
const npv = financial.npv(0.1, cashFlows); // $7,946.11 (positive = good investment)

// Internal Rate of Return - What's the return rate?
const irr = financial.irr(cashFlows); // 16.89%
```

Interest - Various Interest Calculations

```
const interest = new Interest();

// Simple Interest - $1,000 at 5% for 2 years
const simple = interest.simple(1000, 0.05, 2); // $100

// Compound Interest - $1,000 at 5% for 2 years, compounded monthly
const compound = interest.compound(1000, 0.05, 2, 'monthly'); // $104.94

// Effective Annual Rate - What's the real rate with monthly compounding?
const ear = interest.effectiveAnnualRate(0.12, 'monthly'); // 12.68%

// Rule of 72 - How long to double your money at 8%?
const years = interest.ruleOf72(8); // ~9 years
```

Amortization - Loan Schedules

```
const amortization = new Amortization();

// Generate complete loan schedule
const schedule = amortization.schedule({
  principal: 250000,
  annualRate: 0.045,
  termYears: 30,
  extraPayment: 200 // Optional extra monthly payment
});

// Analyze the schedule
console.log(`Monthly Payment: $${schedule.monthlyPayment.toFixed(2)}`);
console.log(`Total Interest: $${schedule.totalInterest.toFixed(2)}`);
console.log(`Payoff in ${schedule.payments.length} months`);

// Each payment includes:
schedule.payments[0]; // {
//   period: 1,
//   payment: 1266.71,
//   principal: 329.21,
//   interest: 937.50,
//   balance: 249670.79,
```

```
//   cumulativePrincipal: 329.21,
//   cumulativeInterest: 937.50
// }
```

Currency - Money Formatting & Conversion

```
const currency = new Currency();

// Set exchange rates
currency.setExchangeRates([
  { from: 'USD', to: 'EUR', rate: 0.85 },
  { from: 'USD', to: 'GBP', rate: 0.73 },
  { from: 'USD', to: 'JPY', rate: 110 }
]);

// Convert currencies
const euros = currency.convert(1000, 'USD', 'EUR'); // €850

// Format currency for display
currency.format(1234.56, 'USD'); // "$1,234.56"
currency.format(1234.56, 'EUR'); // "€1.234,56"
currency.format(1234.56, 'JPY'); // "¥1,235"

// Money operations (ensures same currency)
const total = currency.addMoney(
  currency.money(100, 'USD'),
  currency.money(50, 'USD')
); // $150

// Percentage calculations
currency.percentage(200, 15);           // 30 (15% of 200)
currency.discount(100, 20);             // 80 (20% off)
currency.withTax(100, 0.08);           // 108 (8% tax)
```

Precision and Accuracy

All calculations use decimal.js internally to avoid floating-point precision errors common in JavaScript:

```
// JavaScript floating point error
console.log(0.1 + 0.2); // 0.30000000000000004

// @fin.cx/calculation precision
const calc = new Calculator();
const result = calc.add(0.1, 0.2);
console.log(calc.toString(result)); // "0.3"
```

API Reference

Calculator Options

```
interface ICalculatorOptions {
  precision?: number;    // Decimal places (default: 10)
  rounding?: number;     // Rounding mode (default: ROUND_HALF_UP)
}
```

Compounding Frequencies

- `'annually'` - Once per year
- `'semiannually'` - Twice per year
- `'quarterly'` - Four times per year
- `'monthly'` - Twelve times per year
- `'weekly'` - 52 times per year
- `'daily'` - 365 times per year
- `'continuous'` - Continuous compounding

Currency Options

```
interface ICurrencyOptions {
  code: string;
  symbol?: string;
  decimals?: number;
```

```
  thousandsSeparator?: string;
  decimalSeparator?: string;
  symbolPosition?: 'before' | 'after';
  spaceBetweenSymbolAndValue?: boolean;
}
```

□□ Real-World Examples

Mortgage Calculator

```
import { Financial, Amortization } from '@fin.cx/calculation';

function calculateMortgage(homePrice: number, downPayment: number, rate: number, years:
number) {
  const financial = new Financial();
  const amortization = new Amortization();

  const loanAmount = homePrice - downPayment;
  const monthlyRate = rate / 12;
  const months = years * 12;

  const monthlyPayment = financial.payment(loanAmount, monthlyRate, months);
  const totalPaid = monthlyPayment.mul(months);
  const totalInterest = totalPaid.sub(loanAmount);

  const schedule = amortization.schedule({
    principal: loanAmount,
    annualRate: rate,
    termYears: years
  });

  return {
    monthlyPayment,
    totalInterest,
    schedule
  };
}
```

Investment Analysis

```
import { Financial } from '@fin.cx/calculation';

function analyzeInvestment(initialInvestment: number, cashFlows: number[], discountRate:
number) {
  const financial = new Financial();

  // Add initial investment as negative cash flow
  const allCashFlows = [-initialInvestment, ...cashFlows];

  const npv = financial.npv(discountRate, allCashFlows);
  const irr = financial.irr(allCashFlows);
  const paybackPeriod = calculatePaybackPeriod(allCashFlows);

  return {
    npv: npv.toFixed(2),
    irr: (irr.mul(100)).toFixed(2) + '%',
    profitable: npv.greaterThan(0),
    paybackPeriod
  };
}
```

Multi-Currency Invoice System

```
import { Currency } from '@fin.cx/calculation';

class InvoiceSystem {
  private currency = new Currency();

  constructor() {
    // Set up exchange rates (in production, fetch from API)
    this.currency.setExchangeRates([
      { from: 'USD', to: 'EUR', rate: 0.85 },
      { from: 'USD', to: 'GBP', rate: 0.73 }
    ]);
  }
```

```
  calculateInvoice(items: Array<{price: number, quantity: number}>, currency: string, taxRate:
number) {
    const subtotal = items.reduce((sum, item) =>
      this.currency.add(sum, this.currency.multiply(item.price, item.quantity)),
      0
    );

    const tax = this.currency.tax(subtotal, taxRate);
    const total = this.currency.add(subtotal, tax);

    return {
      subtotal: this.currency.format(subtotal, currency),
      tax: this.currency.format(tax, currency),
      total: this.currency.format(total, currency),
      // Provide total in other currencies
      totalUSD: this.currency.format(this.currency.convert(total, currency, 'USD'), 'USD'),
      totalEUR: this.currency.format(this.currency.convert(total, currency, 'EUR'), 'EUR')
    };
  }
}
```

□□ Configuration

Calculator Precision

```
// Default precision is 10 decimal places
const calc = new Calculator();

// Custom precision for specific use cases
const highPrecision = new Calculator({ precision: 20 });
const moneyCalc = new Calculator({ precision: 4, rounding: Decimal.ROUND_HALF_UP });
```

Custom Currencies

```
const currency = new Currency();

// Register a custom currency
currency.registerCurrency({
  code: 'BTC',
  symbol: '□',
  decimals: 8,
  thousandsSeparator: ',',
  decimalSeparator: '.',
  symbolPosition: 'before'
});

// Use it like any other currency
currency.format(0.00042, 'BTC'); // "□0.00042000"
```

□□ Testing

The package includes comprehensive tests for all calculations:

```
# Run tests
npm test

# Run tests with coverage
npm run test:coverage
```

□□ Acknowledgments

- Built on top of decimal.js for decimal arithmetic
- Inspired by financial calculation needs in modern web applications

License and Legal Information

Please note: The MIT License does not grant permission to use the trade names, trademarks, service marks, or product names of the project, except as required for reasonable and customary use in describing the origin of the work and reproducing the content of the NOTICE file.

Trademarks

This project is owned and maintained by Task Venture Capital GmbH. The names and logos associated with Task Venture Capital GmbH and any related products or services are trademarks of Task Venture Capital GmbH and are not included within the scope of the MIT license granted herein. Use of these trademarks must comply with Task Venture Capital GmbH's Trademark Guidelines, and any usage must be approved in writing by Task Venture Capital GmbH.

Company Information

Task Venture Capital GmbH
Registered at District court Bremen HRB 35230 HB, Germany

For any legal inquiries or if you require further information, please contact us via email at hello@task.vc.

By using this repository, you acknowledge that you have read this section, agree to comply with its terms, and understand that the licensing of the code does not imply endorsement by Task Venture Capital GmbH of any derivative works.

changelog.md for @fin.cx/calculation

[1.0.0] - 2025-07-29

Added

- Initial release of @fin.cx/calculation
- Calculator class with high-precision decimal arithmetic
- Financial class with time value of money calculations (PV, FV, NPV, IRR, XIRR)
- Interest class with simple, compound, and continuous interest calculations
- Amortization class with loan schedules and payment analysis
- Currency class with conversion, formatting, and monetary operations
- Comprehensive test suite
- Full TypeScript support
- CI/CD workflows for automated testing and publishing