

Modular Arithmetic

[Read this note online](#)

Modular arithmetic, sometimes called 'clock arithmetic', is a system of arithmetic for integers in which numbers wrap around after reaching a fixed value called the modulus. If you take 15 hours past midnight on a 12-hour clock, you don't land at 15 o'clock, you land at 3 o'clock, because clocks wrap around every 12 hours. That's modular arithmetic. Formalized by Gauss in *Disquisitiones Arithmeticae* (1801), it's the language of cryptography, hash functions, digital signal processing, and much of computer science.

Modular arithmetic preserves congruence under addition and multiplication. Division needs an extra condition because a divisor may lack an inverse.



modular arithmetic

$$15 \bmod 12 = 3$$

$$17 \bmod 5 = 2$$

3		2
5		

Modular arithmetic wraps numbers around a modulus, like hours on a clock face.

Definition

Two integers a and b are **congruent modulo n** (written $a \equiv b \pmod{n}$) if their difference is divisible by n . In symbols:

$$a \equiv b \pmod{n} \iff n \mid (a - b)$$

Equivalently, a and b leave the same remainder when divided by n . The number n is the *modulus*.

Examples. $17 \equiv 5 \pmod{12}$ because $17 - 5 = 12$ is divisible by 12. $38 \equiv 3 \pmod{7}$ because $38 = 5 \cdot 7 + 3$. $-1 \equiv 11 \pmod{12}$ because $-1 - 11 = -12$ is divisible by 12.

Operations Mod N

Addition, subtraction, and multiplication all work normally modulo n :

- $(a + b) \pmod{n} = ((a \pmod{n}) + (b \pmod{n})) \pmod{n}$.
- $(a - b) \pmod{n}$, same rule.
- $(a \cdot b) \pmod{n} = ((a \pmod{n}) \cdot (b \pmod{n})) \pmod{n}$.
- **Division is trickier:** you can only divide by b modulo n when $\gcd(b, n) = 1$, and division is done by finding the *modular inverse* of b .

Worked Examples

Day-of-week calculation. Today is Wednesday. What day is it 100 days from now? Take $100 \pmod{7} = 2$. Add 2 days to Wednesday: Friday.

Last digit. The last digit of 7^{100} is $7^{100} \pmod{10}$. Powers of 7 mod 10 cycle: 7, 9, 3, 1, 7, 9, 3, 1, ... with period 4. $100 \pmod{4} = 0$, so $7^{100} \equiv 7^4 \equiv 1 \pmod{10}$. Last digit is 1.

Solving a congruence. Solve $3x \equiv 7 \pmod{11}$. The inverse of 3 mod 11 is 4 (because $3 \cdot 4 = 12 \equiv 1 \pmod{11}$). So $x \equiv 4 \cdot 7 = 28 \equiv 6 \pmod{11}$.

Fermat's Little Theorem

If p is prime and a is not divisible by p :

$$a^{p-1} \equiv 1 \pmod{p}$$

Equivalently, $a^p \equiv a \pmod{p}$ for all integers a (including those divisible by p). This is a workhorse

for computing very large powers modulo a prime and is the basis of many primality tests.

Example. Compute $2^{100} \bmod 13$. By Fermat's little theorem, $2^{12} \equiv 1 \pmod{13}$. And $100 = 8 \cdot 12 + 4$, so $2^{100} = (2^{12})^8 \cdot 2^4 \equiv 1 \cdot 16 \equiv 3 \pmod{13}$.

Applications

- **RSA encryption.** RSA computes ciphertext as $\text{message}^e \bmod n$, where $n = pq$ is the product of two large primes and e is the public exponent. Decryption uses $\text{message}^d \bmod n$, where d is the private exponent. Everything happens in modular arithmetic.
- **Hash functions.** Hash tables use $h(\text{key}) \bmod \text{table_size}$ to map keys to bucket indices. Prime table sizes minimize clustering.
- **Check digits.** ISBN-10 check digit uses arithmetic mod 11; credit card numbers (Luhn algorithm) use a mod 10 check.
- **Pseudo-random number generators.** Linear congruential generators (LCGs) compute $X_{n+1} = (a \cdot X_n + c) \bmod m$. Choice of a , c , m determines the quality of the output sequence.
- **Cyclic structures.** Days of the week, hours, calendars, angles ($\bmod 360^\circ$), musical pitches ($\bmod 12$), anything that repeats periodically is naturally described with modular arithmetic.
- **Error-correcting codes.** Reed-Solomon and CRC checksums operate in modular arithmetic (often over finite fields with prime modulus).

Related study notes: [Prime Numbers](#), Euclidean Algorithm, Cryptography, [Fundamental Theorem of Algebra](#).

The related [OpenStax mathematics textbooks](#) provide additional explanations and exercises for the mathematical definitions used here.

When Division Is Allowed

Solve $3x \equiv 4 \pmod{7}$. Since $3 \times 5 \equiv 1 \pmod{7}$, multiplying both sides by 5 gives $x \equiv 20 \equiv 6 \pmod{7}$. Substitution checks it: $3 \times 6 = 18$, which has remainder 4.

Now consider $2x \equiv 1 \pmod{6}$. No solution exists, because $2x$ is even while an integer congruent to 1 modulo 6 is odd. The number 2 has no multiplicative inverse modulo 6. Canceling it as if the congruence were an ordinary real equation would lose that obstruction.

The points to check in this example are:

- A divisor has a modular inverse when it is coprime to the modulus.
- Negative integers can be represented by their nonnegative remainder.

- Congruent values need not be equal as ordinary integers.

Check Your Understanding

Try the questions before reading the answers.

Question 1. What is -3 modulo 7 in the usual nonnegative convention?

Answer. 4.

Question 2. What is the inverse of 2 modulo 5 ?

Answer. 3, because 2×3 has remainder 1 modulo 5.

Practice Questions

Work each question before reading its solution. The set runs from direct recall and substitution to the applied questions that exams actually use to separate grades. The downloadable PDF includes these questions and their solutions.

Question 1. Compute $17 \bmod 5$, $100 \bmod 7$, and $-3 \bmod 8$.

Solution. $17 = 3 \times 5 + 2$, so 2. $100 = 14 \times 7 + 2$, so 2. For -3 : add 8 to reach the standard range, giving 5, since $-3 = -1 \times 8 + 5$. The result always lands in 0 to $m - 1$, negatives included.

Question 2. It is 9 o'clock. What time does a 5-hour meeting end, and what computation is that?

Solution. $9 + 5 = 14 \equiv 2 \pmod{12}$: it ends at 2 o'clock. Clock time is arithmetic modulo 12; days of the week are modulo 7. You have used modular arithmetic your whole life without the notation.

Question 3. What does $a \equiv b \pmod{m}$ mean? Give the 2 equivalent formulations.

Solution. Either: a and b leave the same remainder on division by m . Or: m divides $a - b$. The second is usually easier to use in proofs: $37 \equiv 2 \pmod{5}$ because $5 \mid 35$.

Question 4. Find the last digit of 7^{100} .

Solution. Last digits are mod 10. Powers of 7 cycle: 7, 9, 3, 1, 7, 9, ... with period 4. Since $100 = 4 \times 25$, the exponent lands at the cycle's end: last digit 1. Huge powers collapse to short cycles: that is the whole trick.

Question 5. Compute $3^{50} \bmod 7$ without ever holding a large number.

Solution. Reduce as you go. $3^6 \equiv 1 \pmod{7}$ (Fermat's little theorem, or check: $3^2 = 2$, $3^3 = 6$, $3^6 = 6^2 = 36 \equiv 1$). Then $3^{50} = 3^{48} \cdot 3^2 = (3^6)^8 \cdot 9 \equiv 1 \cdot 2 = 2 \pmod{7}$.

Question 6. Prove the divisibility-by-9 test: a number is divisible by 9 exactly when its digit sum is.

Solution. Since $10 \equiv 1 \pmod{9}$, every power $10^k \equiv 1$. A number $d_n \cdots d_1 d_0 = \sum d_k 10^k \equiv \sum d_k \pmod{9}$: the number and its digit sum are congruent, so one is divisible by 9 exactly when the other is. The digit test is modular arithmetic in costume.

Question 7. Solve $x + 7 \equiv 3 \pmod{10}$ and then $3x \equiv 4 \pmod{7}$.

Solution. First: $x \equiv 3 - 7 = -4 \equiv 6 \pmod{10}$. Second: multiply by the inverse of 3 mod 7, which is 5 (because $15 \equiv 1$): $x \equiv 5 \times 4 = 20 \equiv 6 \pmod{7}$. Division works only through inverses, and an inverse exists only when gcd of the coefficient and modulus is 1.

Question 8. Why does an ISBN-10 check digit use arithmetic mod 11, and what does it catch?

Solution. The check digit makes $\sum_{i=1}^{10} i \cdot d_i \equiv 0 \pmod{11}$. Because 11 is prime, every single-digit error and every adjacent-digit swap changes the weighted sum by a nonzero amount mod 11, so both error types are always detected. Composite moduli would let some errors slip through.

Question 9. Compute $2^{10} \bmod 1000$, then explain what "the last 3 digits of 2^n " means modularly.

Solution. $2^{10} = 1024 \equiv 24 \pmod{1000}$. The last 3 digits of any number are exactly its residue mod 1000, just as the last digit is its residue mod 10. Digit questions are always modulus questions.

Question 10. Where does modular arithmetic run in your daily life? Name 3 systems.

Solution. RSA encryption performs its encryption and decryption as modular exponentiation with 600-digit moduli, securing effectively every HTTPS connection. Hash tables place keys in buckets by hashing mod the table size. Check digits on ISBNs, credit cards (Luhn, mod 10), and bank codes catch typos. Clocks, calendars, and cyclic schedules round out the list.

Frequently Asked Questions

What is modular arithmetic?

A system of arithmetic for integers in which numbers wrap around after reaching a fixed modulus n . Two numbers are 'congruent mod n ' if they leave the same remainder when divided by n . The most familiar example is a clock: 14 o'clock is the same as 2 o'clock, because they're congruent mod 12.

What does ' $a \equiv b \bmod n$ ' mean?

It means ' a and b leave the same remainder when divided by n ', or equivalently, ' $a - b$ is divisible by n '. Example: $17 \equiv 5 \pmod{12}$ because both 17 and 5 leave remainder 5 when divided by 12, and $17 - 5 = 12$ is divisible by 12.

Can you divide in modular arithmetic?

Not always. You can divide by $b \bmod n$ only if b and n are coprime ($\gcd(b, n) = 1$). When they are, division by b is performed by multiplying by the modular inverse of b , the number b^{-1} such that $b \cdot b^{-1} \equiv 1 \pmod{n}$. The extended Euclidean algorithm finds the inverse efficiently.

What is Fermat's Little Theorem?

If p is prime and a is not divisible by p , then $a^{p-1} \equiv 1 \pmod{p}$. This is a fast way to compute large powers modulo a prime, and it's the basis of several primality tests, including the Miller-Rabin probabilistic test. It also underlies parts of RSA cryptography.

How is modular arithmetic used in cryptography?

It's the entire mathematical setting of RSA and many other cryptosystems. RSA computes ciphertext = $\text{message}^e \bmod n$, where n is the product of two large primes. Without modular arithmetic, encryption and decryption, done via modular exponentiation, wouldn't make sense or be feasible at scale.

Who invented modular arithmetic?

Carl Friedrich Gauss formalized it in 'Disquisitiones Arithmeticae' (1801), introducing the $\equiv$ notation and laying down the basic theorems. The underlying ideas are much older, Chinese mathematicians had results equivalent to the Chinese Remainder Theorem by the 3rd century CE, and Euclid's algorithm goes back to about 300 BCE.