Arithmetic, relational and logical operators

1.2.3 · Computational thinking · revision notes

Operators do the calculations and comparisons in every algorithm: arithmetic operators, relational (comparison) operators and the logical operators AND, OR and NOT. Questions ask you to evaluate expressions, especially ones with integer division and modulus, to decide whether conditions are True or False, and to write conditions of your own.

Key facts

- + add, − subtract, * multiply, / divide (real result: 7 / 2 = 3.5)
- // integer division: 17 // 5 = 3
- % modulus (remainder): 17 % 5 = 2
- ** exponentiation: 2 ** 4 = 16
- == equal to, != not equal to, < less than, > greater than, <= less than or equal to, >= greater than or equal to
- and: True only if both are True. or: True if at least one is True. not: reverses
- Precedence: brackets, then **, then * / // %, then + −
- n % 2 == 0 means n is even

Notes

Arithmetic operators

- `+` add, `-` subtract, `*` multiply, `/` divide. In Python, `/` always gives a real (float) result: `7 / 2` is 3.5 and `8 / 2` is 4.0.
- **Integer division** `//` gives the whole number of times one number goes into another: `17 // 5` is 3.
- **Modulus** `%` gives the remainder: `17 % 5` is 2.



3 groups of 5: 17 // 5 = 3 2 left: 17 % 5 = 2

// counts the whole groups; % gives what is left over

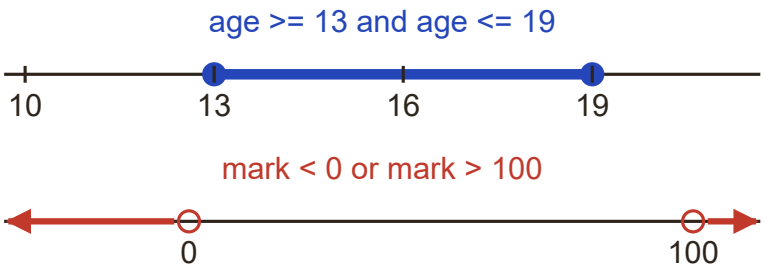
- **Exponentiation** `**` raises to a power: `2 ** 4` is 16.
- Uses: `n % 2 == 0` is True when n is even; `n % 10` is the last digit of n; 135 minutes is `135 // 60` = 2 hours and `135 % 60` = 15 minutes.
- Order: brackets, then `**`, then `* / // %` (left to right), then `+` and `-` (left to right). So `3 + 2 ** 3` is 11.

Relational operators

- They compare two values and give a Boolean result: True or False.
- `==` equal to, `!=` not equal to, `<` less than, `>` greater than, `<=` less than or equal to, `>=` greater than or equal to.
- `=` assigns a value; `==` compares two values.
- Translate words carefully: 'at least 18' is `>= 18`; 'over 18' is `> 18`; 'no more than 50' is `<= 50`.

Logical operators

- **AND** (`and`): True only when both conditions are True.
- **OR** (`or`): True when at least one condition is True.
- **NOT** (`not`): reverses a condition, so True becomes False and False becomes True.
- In range: `age >= 13 and age <= 19`. Out of range: `mark < 0 or mark > 100`.



and joins two tests to make a range between; or catches values outside at either end. A filled circle includes the value, an open circle leaves it out

- Without brackets, `not` is done first, then `and`, then `or`. Use brackets to make the meaning clear, e.g. `(day == "Sat" or day == "Sun") and not raining`. Grade 7+ only

How to answer each type of question

Evaluate arithmetic expressions

1 mark each | Grade 5

1. Do brackets and `**` first, then `*`, `/`, `//` and `%`, then `+` and `-`.
2. For `//` give only the whole-number part; for `%` give only the remainder.
3. Remember that `/` gives a real number in Python, e.g. 4.0 not 4.

Example. State the result of each expression.

(a) `23 // 4`

(b) `23 % 4`

(c) `3 + 2 ** 3`

(d) `10 / 4`

Model answer.

(a) 5 (1)

(b) 3 (1)

(c) 11 (1), because `2 ** 3` = 8 is worked out first

(d) 2.5 (1)

Decide whether a condition is True or False

1 mark each | Grade 6

1. Replace each variable with its value.
2. Work out each comparison to True or False.
3. Combine them: brackets first, then not, then and, then or.

Example. `x = 8` and `y = 3`. State whether each condition is True or False.

(a) `x > 5 and y > 5`

(b) `not (x == y)`

(c) `x % y == 2 or y > x`

Model answer.

(a) False (1), because `y > 5` is False

(b) True (1), because `x == y` is False and not False is True

(c) True (1), because 8 % 3 is 2, so the first part is True

Write a condition

2 to 4 marks | **Grade 7**

1. Turn each part of the rule into a comparison, choosing `<` or `<=` carefully at the boundaries.
2. Join the parts with and / or, and use brackets if you mix them.
3. Test it with a value just inside and a value just outside the range.

Example. A code is valid if it is between 1000 and 9999 inclusive and it is an even number.
Write a Python `if` statement that outputs `Valid` when the integer variable `code` holds a valid code.

Model answer.

```
if code >= 1000 and code <= 9999 and code % 2 == 0:
    print("Valid")
```

Both boundary checks, `>= 1000` and `<= 9999` (1); tests for an even number with `code % 2 == 0` (1); all three joined with and in an if statement that outputs Valid (1).

Use // and % to solve a problem

2 to 3 marks | **Grade 7**

1. Use `//` for 'how many whole ...'.
2. Use `%` for 'how many are left over'.
3. Check your code with the example numbers in the question.

Example. A program inputs a whole number of minutes, e.g. 135.
Write Python code that outputs the number of whole hours and the number of minutes left over, e.g. `2 hours 15 minutes`.

Model answer.

```
mins = int(input("Enter minutes: "))
hours = mins // 60
left = mins % 60
print(hours, "hours", left, "minutes")
```

Inputs the minutes as an integer (1); uses `// 60` to find the whole hours (1); uses `% 60` to find the minutes left over and outputs both (1).

Don't lose marks

- Using `=` instead of `==` in a condition.
- Writing `x > 5 and < 10`. Each side of and needs a full comparison: `x > 5 and x < 10`.
- Using and for an out-of-range check: `mark < 0 and mark > 100` can never be True.
- Getting boundaries wrong: 'at least 18' needs `>=`, not `>`.
- Forgetting that `**` is worked out before `*` and `+`, or that `/` gives a real number.
- Mixing up `//` (whole number of times) and `%` (remainder).

More tips

Memory tricks

- 17 ÷ 5 = 3 remainder 2: integer division (`//`) gives the 3, modulus (`%`) gives the 2.
- Even test: `n % 2 == 0`. Last digit: `n % 10`. Remove the last digit: `n // 10`.
- One equals sign sets a value; two equals signs check it.
- In range uses **and**. Out of range uses **or**, because a value cannot be too low and too high at once.

Exam technique

- Underline the boundary words in the question (at least, more than, up to, inclusive) and choose the operator for each one.
- Write down the value of each part of a condition, such as 8 % 3 = 2, so that you can check it.
- Write logical operators the way the question does: AND, OR, NOT in a written answer, or `and`, `or`, `not` in Python.

What each grade needs

What you need to be able to do, from the first marks up to the top grade.

Grade 3 — **Use the four basic arithmetic operators**
Add, subtract, multiply and divide, knowing that `/` gives a real number.

Grade 4 — **Use relational operators to compare values**
Use `==`, `!=`, `<`, `>`, `<=` and `>=` to give True or False.

Grade 5 — **Calculate with modulus and integer division**
`17 // 5` is 3 and `17 % 5` is 2.

Grade 5 — **Combine conditions with AND, OR and NOT**
Join comparisons to test a range or several conditions at once.

Grade 6 — **Apply operator precedence to evaluate expressions**
Brackets first, then exponentiation, then multiplication and division, then addition and subtraction.

Grade 7 — **Use modulus and integer division to solve problems**
E.g. test whether a number is even, or convert minutes into hours and minutes.

Grade 8 — **Write complex conditions with brackets and NOT**
Build conditions such as `not (x == 0 or y == 0)` that do exactly what is needed.

Free notes, Full packs, practice papers and mock exams at chhetriacademy.com. Written for this site; not taken from past papers. Stuck? 1-to-1 tutoring with a free 30-minute trial lesson.