Developing code: readability, errors, validation and testing

6.1 · Problem solving with programming · revision notes

Writing code that works and that other people can follow: making programs readable, finding and fixing syntax, logic and runtime errors, validating input, authenticating users, and testing with normal, boundary and erroneous data. It is central to Paper 2, the onscreen programming exam, where you correct, improve and test Python code.

Key facts

- Syntax error: breaks the rules of the language, so the program will not run
- Runtime error: the program crashes while running, e.g. division by zero
- Validation checks: presence, length, range, pattern
- Authentication: confirming a user's identity, e.g. looking up an ID and checking its password
- Efficiency: number of comparisons, passes through loops, memory used
- Logic error: the program runs but gives the wrong result
- Readability: comments, meaningful identifiers, indentation, white space
- Validation checks data is sensible; it cannot prove data is correct
- Normal data: valid, accepted. Boundary data: on the edge of the range, accepted. Erroneous data: invalid, rejected
- Decomposition: break a problem into parts. Abstraction: remove unnecessary detail

Notes

Readable code

- **Comments** (after `#`) explain what the code does. Python ignores them.
- **Meaningful identifiers**, e.g. `totalCost` not `t`, show what each variable or subprogram is for.
- **Indentation** shows which lines are inside an `if`, a loop or a subprogram. In Python it is part of the syntax.
- **White space**: blank lines between sections and spaces around operators. Readable code is quicker to debug and **maintain**.

Three types of error

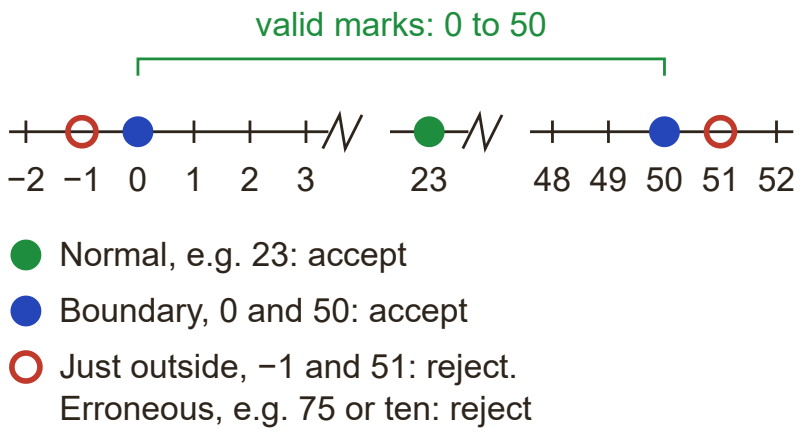
- **Syntax error**: the code breaks the rules of the language, so it cannot be translated and will not run, e.g. a missing colon, bracket or quotation mark.
- **Logic error**: the program runs but gives the wrong result, e.g. `>` instead of `>=`.
- **Runtime error**: the program crashes while running, e.g. dividing by zero, `int("ten")` or an index that is not in the list.
- Find logic errors with a trace table, or with temporary `print()` lines that show the values of variables.

Validation and authentication

- **Validation** checks that input is sensible and allowed before it is used. It cannot prove data is correct: 41 is a valid age even if the user is 14.
- **Presence check**: something was entered. **Length check**: an allowed number of characters. **Range check**: a value between limits. **Pattern check** (format check): the right form, e.g. two letters then four digits.
- **Authentication** confirms that users are who they say they are: look up the user's ID in the stored data and check that the password stored with it matches.

Testing and evaluating

- **Normal data**: valid values inside the range, e.g. 23 for a mark from 0 to 50. They should be accepted.
- **Boundary data**: valid values on the edges of the range (0 and 50). Also test just outside (−1 and 51): these must be rejected.
- **Erroneous data**: invalid data that should be rejected, e.g. 75 or `ten`.



Test data for a mark from 0 to 50 (the axis is broken between the ends).

- A **test plan** lists each test's data, type and expected result. A program is **fit for purpose** if it meets all of its requirements.
- Judge **efficiency** by the number of comparisons, passes through loops and memory used. A search that stops when it finds the item usually makes fewer passes than one that checks every item. Grade 8+ only

From problem to program

- **Decomposition**: split the problem into parts, e.g. one subprogram per task. **Abstraction**: keep only the details the program needs.
- Convert a flowchart or pseudocode step by step: inputs become `input()`, processes become assignments, decisions become `if` statements or loop conditions, outputs become `print()`.
- **Refine** after testing: fix errors and replace repeated code with a loop or subprogram.

How to answer each type of question

Identify the validation check

1 mark each | Grade 4

1. Look at what the condition tests: whether anything was entered, the number of characters, the size of a number, or the form of the data.
2. Name the check exactly: presence, length, range or pattern.

Example. Each line of code below is the start of a validation check.

```
01 if username == "":
02 if len(postcode) < 6 or len(postcode) > 8:
03 if quantity < 1 or quantity > 20:
04 if not phone.isdigit():
```

State the type of validation check carried out on each line. (4)

Model answer.

01: presence check	(1)
02: length check	(1)
03: range check	(1)
04: pattern check: only digits are allowed	(1)

Give ways to improve readability

2 to 4 marks | Grade 4

1. Choose techniques that fit the code you are given: comments, meaningful identifiers, indentation, white space.
2. Link each technique to this program, e.g. name the variable you would rename.
3. Explain how it helps a programmer who reads or maintains the code.

Example. This program works out the cost of a meal, including a 10% service charge. It works, but it is hard to read.

```
a=float(input("?"))
b=a*0.1
print(a+b)
```

Give **two** ways to make this program easier to read. Explain how each way helps a programmer. (4)

Model answer.

Use meaningful identifiers, e.g. `mealCost` and `serviceCharge` instead of `a` and `b` (1), so it is clear what each variable stores (1).

Add a comment such as `# add a 10% service charge` (1), which explains the purpose of the calculation to anyone maintaining the code (1).

Also accept white space, e.g. spaces around `=` and `*`, with a matching explanation.

Complete a test plan

3 to 6 marks **Grade 5**

1. Find the valid range or rule in the question.
2. Normal: a value well inside the range. Boundary: a value exactly on a limit. Erroneous: a value that breaks the rule.
3. Give an actual value, not a description, and state exactly what should happen: accepted, or rejected with a message.

Example. A program asks how many tickets a customer wants. Valid values are whole numbers from 1 to 6 inclusive.
Complete the test plan by giving test data and the expected result for each type of test. (3)

Type of test	Test data	Expected result
Normal		
Boundary		
Erroneous		

Model answer.

Type of test	Test data	Expected result
Normal	e.g. 3	Accepted (1)
Boundary	1 or 6	Accepted (1)
Erroneous	e.g. 9, 0 or `two`	Rejected with an error message (1)

Each mark needs both the test data and the correct expected result.

Identify and correct errors in code

2 to 4 marks **Grade 6**

1. Read what the program should do, then read the code line by line.
2. Look for syntax errors first: missing colons, brackets or quotation marks, and wrong indentation.
3. Trace the code with simple test values to find logic errors: compare what it does with what it should do.
4. Give the line number, name the type of error and write out the whole corrected line.

Example. This program should input three marks and output their mean. It contains one syntax error and one logic error.

```
01 total = 0
02 for count in range(3)
03     mark = int(input("Enter a mark: "))
04     total = total + mark
05 mean = total / 2
06 print("Mean:", mean)
```

(a) Give the line number of the syntax error and write the corrected line. (2)
(b) Give the line number of the logic error and write the corrected line. (2)

Model answer.

(a) Line 02 (1): `for count in range(3):` (1)

(b) Line 05 (1): `mean = total / 3` (1)

Explain which program is more efficient 2 to 4 marks **Grade 8**

1. Count what each program does: comparisons made, passes through each loop, and extra variables or lists stored.
2. Say which program is more efficient.
3. Explain why, comparing the two programs directly (use 'whereas').

Example. Both programs set `found` to `True` if `target` is in the list `names`, which holds 500 names.

Program A

```
found = False
for name in names:
    if name == target:
        found = True
```

Program B

```
found = False
index = 0
while found == False and index < len(names):
    if names[index] == target:
        found = True
    index = index + 1
```

Explain which program is more efficient. (3)

Model answer.

Program B (1). It stops looping as soon as the target is found (1), whereas Program A always makes 500 passes through the loop and 500 comparisons, even when the target is the first name (1).

(If the target is not in the list, both programs make 500 passes.)

Don’t lose marks

- Saying validation makes sure data is correct. It only checks that data is reasonable and follows the rules.
- Choosing boundary data from the middle of the range. Boundary values sit exactly on the limits.
- Calling a crash a logic error. A program that stops part-way with an error message has a runtime error.
- Writing a description such as 'a big number' in a test plan instead of an actual value such as 51.
- Correcting one error but also changing other lines, e.g. their indentation, which adds a new error.
- Saying comments make a program run better. They are ignored when the program runs and only help people.

More tips

Memory tricks

- Syntax stops it starting, runtime stops it running, logic lets it finish with the wrong answer.
- Validation checks in four questions: is it there (presence)? Is it the right size (length)? Is it within the limits (range)? Is it the right shape (pattern)?
- Boundary tests for a whole-number range from a to b: test a and b (accepted), then a − 1 and b + 1 (rejected).
- Readability checklist: comments, identifiers, indentation, white space.
- Validation asks 'is this data reasonable?'; authentication asks 'is this really you?'

Exam technique

- For 'identify the error', give the line number and the type of error; for 'correct the error', write out the whole corrected line.
- State expected results precisely, e.g. 'rejected, error message shown and asked again', not 'it works'.
- In the onscreen exam, run your program after every change and test it with normal, boundary and erroneous data.
- When improving readability, do not change what the program does, and rename a variable everywhere it appears.
- Use the exact terms: syntax, logic, runtime; presence, length, range, pattern; normal, boundary, erroneous.

What each grade needs

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

Grade 3 — **Name the three types of program error**
Syntax, logic and runtime errors, with an example of each.

Grade 4 — **Make code easier to read and maintain**
Add comments, meaningful identifiers, indentation and white space without changing what the program does.

Grade 4 — **Identify presence, length, range and pattern checks**
Say which validation check a line of code carries out and why it is needed.

Grade 5 — **Choose normal, boundary and erroneous test data**
Pick actual values for a test plan and give the expected result of each test.

Grade 6 — **Locate and correct errors in a program**
Find the line with the error, name the type of error and write the corrected line.

Grade 6 — **Convert a flowchart or pseudocode into Python**
Turn each input, process, decision and output of an algorithm into the matching Python statement.

Grade 7 — **Write validation and authentication code**
Repeat an input until it passes every check, and check an ID and password against stored data.

Grade 8 — **Evaluate fitness for purpose and efficiency**
Use test results and the number of comparisons, loop passes and memory used to judge a program.

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.