Defensive design

2.3.1 · Algorithms and programming (Paper 2) · revision notes

Defensive design means planning a program so that it copes with misuse, only lets the right people in, checks its inputs and is easy for other programmers to maintain. Expect questions on validation checks, authentication, maintainability and writing validation code.

Key facts

- Anticipating misuse: plan for unexpected input and attempts to break or break into the program
- Validation: checks input is sensible or allowed before it is used (not that it is correct)
- Length check: number of characters; presence check: not blank
- Maintainability: sub programs, meaningful names, indentation, comments
- Authentication: confirming who the user is (username and password, limited attempts)
- Range check: between limits; type check: correct data type
- Format check: follows a pattern, such as DD/MM/YYYY

Notes

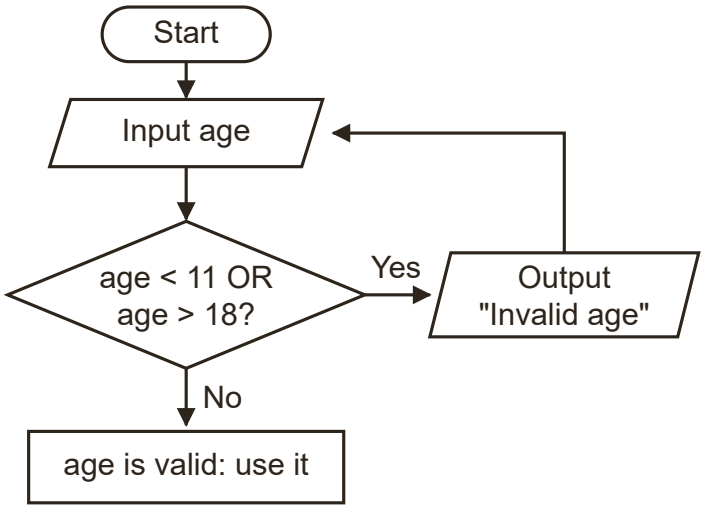
Anticipating misuse and authentication

- **Anticipating misuse**: thinking ahead about how users might enter unexpected data, or try to break into or crash the program, and designing the program to cope. Examples: blank inputs, text where a number is expected, very large values, dividing by zero, trying passwords over and over.
- **Authentication**: confirming the identity of a user before they get access, for example with a username and password.
- Authentication can be made stronger by limiting the number of login attempts, locking the account after several failures, and not saying which of the username or password was wrong.

Input validation

- **Validation** checks that input data is sensible and follows the rules before the program uses it.
- **Range check**: the value is between set limits (an age from 11 to 18).
- **Type check**: the data is the correct data type (a whole number, not text).
- **Length check**: the number of characters is allowed (a password of at least 8 characters).
- **Presence check**: something has been entered (the field is not blank).
- **Format check**: the data follows a pattern (a date written as DD/MM/YYYY).
- Validation cannot check the data is correct: a valid but wrong date of birth is still accepted.
- Example:

```
age = int(input("Enter age (11 to 18)"))
while age < 11 OR age > 18
   print("Invalid age")
   age = int(input("Enter age (11 to 18)"))
endwhile
```



The validation loop in the example, as a flowchart

Maintainability

- Maintainable code is easy for another programmer (or you, later) to understand, fix and update.
- **Sub programs** split the code into small, named parts that can be understood and tested one at a time.
- **Naming conventions**: meaningful identifiers, such as totalCost instead of t, show what each variable holds.
- **Indentation** shows which lines are inside each loop or if statement.
- **Comments** explain what sections of code do and why.

How to answer each type of question

Identify and describe validation checks

2 to 4 marks | Grade 4

1. Name the check that fits the data.
2. Describe exactly what it checks, using the values in the question.

Example. A sign-up form asks for a username, which must be 6 to 12 characters long, and an age, which must be from 13 to 99.
Identify a suitable validation check for each input and describe what it checks. [4]

Model answer.

Username: length check (1), which checks the username has from 6 to 12 characters (1).
Age: range check (1), which checks the age is from 13 to 99 (1).

Write validation code

3 to 4 marks | Grade 6

1. Input the value.
2. Use a loop that repeats while the value is invalid (or until it is valid).
3. Input the value again inside the loop, with an error message if asked.
4. Check the condition includes the limits correctly.

Example. Write an algorithm that asks the user to enter a quantity, and keeps asking until a whole number from 1 to 20 is entered. [3]

Model answer.

```
do
   qty = int(input("Enter quantity (1 to 20)"))
until qty >= 1 AND qty <= 20
```

A loop that repeats until the input is valid (1); the input is inside the loop (1); the condition accepts only 1 to 20 inclusive (1).

Improve the maintainability of code

2 to 4 marks | Grade 5

1. Look for single-letter names, no indentation and no comments.
2. State an improvement that applies to this code.
3. Explain how it helps another programmer.

Example. Give **two** ways the maintainability of this code could be improved, and explain how each helps.

```
a = 0
for b = 1 to 10
c = int(input("?"))
a = a + c
next b
print(a / 10)
```

[4]

Model answer.

Use meaningful variable names, such as total instead of a (1), so the purpose of each variable is clear (1).

Indent the lines inside the loop (1) so it is clear which lines are repeated (1).

Also accept: add comments (1) to explain what the code does (1).

Explain how to reduce misuse

2 to 4 marks | Grade 7

1. Give a design feature, such as limiting login attempts or validating inputs.
2. Explain what misuse it prevents.
3. Link it to the scenario.

Example. An online game requires players to log in. Describe **two** ways the login system could be designed to reduce misuse. [4]

Model answer.

Limit the number of password attempts, locking the account after, say, three failures (1), which stops someone guessing passwords over and over (1).

Validate every input, such as checking the username is not blank and not too long (1), so unexpected data cannot cause errors or crash the program (1).

Don’t lose marks

- ✗ Naming a check without describing it with the values from the question.
- ✗ Choosing a range check for text, or a length check for a number range.
- ✗ Writing validation with an if statement only, so the user is never asked again.

- Getting the loop condition backwards (repeating while the data is valid).
- Saying comments make the program run better: they only help people read the code.

More tips

Memory tricks

- Validation checks: 'Really Tidy Little Pupils Follow' = Range, Type, Length, Presence, Format.
- Maintainability: 'SNIC' = Sub programs, Naming, Indentation, Comments.
- Valid is not the same as correct: validation checks the data is sensible, not true.

Exam technique

- When a check is asked for, name it AND describe it for this data.
- In validation code, test your condition with a value just inside and just outside each limit.
- For maintainability, the four OCR terms are sub programs, naming conventions, indentation and commenting: use them.

What each grade needs

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

Grade 3 — **Name types of validation check**
Range, type, length, presence and format checks.

Grade 4 — **Choose a suitable validation check**
Match the check to the data, such as a range check for an age.

Grade 5 — **Describe ways to make code maintainable**
Sub programs, meaningful names (naming conventions), indentation and comments.

Grade 5 — **Explain what authentication is**
Checking that users are who they say they are, for example with a username and password.

Grade 6 — **Write validation code with a loop**
Keep asking for the input until it meets the rule, giving a message if it is invalid.

Grade 7 — **Explain how to anticipate misuse**
Plan for unexpected input, repeated login attempts and people trying to break the program.

Grade 8 — **Explain the limits of validation**
Validation checks data is sensible, not that it is correct.

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.