



## Robust Programs

PDL Name

Date

---

---

---

---

---

---

---

## Key Information

- 1) Remember this booster is here to **help you**. Please consider your behaviour in the chat.
- 2) If you are in a room with a teacher/group, please login to the meeting. This is so we can mark your attendance. This information goes into a **prize draw**.
- 3) Make sure the name on the meeting is the **SAME** as the name on your Isaac account. We can't mark you present if they don't match.



---

---

---

---

---

---

---

## Starter

2 minutes

- What is the purpose of input validation, and why is it important?

```
user_input = input("Enter your age: ")
if user_input.isdigit() and int(user_input) > 0:
    print("Valid age!")
else:
    print("Invalid input. Please enter a positive number.")
```

- Answers in the chat



---

---

---

---

---

---

---

## Answers

### Purpose of Input Validation

- Ensures data entered is valid and within expected limits.
- Prevents errors that may cause the program to crash.

### Why It's Important

- Improves program reliability and user experience.
- Enhances security by reducing invalid or malicious inputs.




---

---

---

---

---

---

---

---

## What was the point of the starter?

- This question highlights the importance of input validation in programming.
- It prepares you to write code that handles user inputs effectively and avoids errors.

**Input Validation** – Ensures data meets expected criteria before processing.

**Range Checks** – Confirms values fall within specific limits (e.g. age > 0).

**Presence Checks** – Verifies required data is not left blank.




---

---

---

---

---

---

---

---

## Intended learning outcomes

- Be able to explain the importance of defensive design in creating robust programs.
- Be able to implement input validation techniques to ensure data integrity.
- Be able to write simple authentication routines (e.g. username and password checks).
- Be able to apply techniques for writing maintainable code, including naming conventions, subprograms, and commenting.




---

---

---

---

---

---

---

---

## Check in

1 minute

On a scale of 1–10 (1 = low, 10 = high), rate your confidence in:

1. Understanding what defensive design is and why it's important.
2. Writing basic input validation routines to check user input.
3. Creating maintainable code using clear comments and naming conventions.

Post your answers in the chat, e.g. 1(5), 2(7), 3(4)

---

---

---

---

---

---

---

---

## What is Defensive Design?

- Defensive Design is about writing code to:
  - Prevent misuse of the program.
  - Ensure the program works as intended, even with unexpected inputs.
- It helps make software robust and reliable.

---

---

---

---

---

---

---

---

## Why is Defensive Design Important?

- Prevents program failures caused by misuse.
- Reduces bugs and unexpected behaviour.
- Protects systems from invalid or malicious inputs.
- Ensures user confidence by improving reliability.

---

---

---

---

---

---

---

---

## Anticipating Misuse

- Plan for how users might misuse the program.
- Example: A user enters text instead of a number.
- Solutions:
  - Use input validation to check data types, lengths, or ranges.
  - Provide error messages to guide users.



---

---

---

---

---

---

---

## Example



```
user_input = input("Enter your age: ")
if user_input.isdigit():
    print("Valid input")
else:
    print("Invalid input")
```



---

---

---

---

---

---

---

## Authentication

- Authentication ensures only authorised users can access the system.
- Common methods:
  - Username and password checks.
  - Multi-factor authentication
  - e.g. One Time Passwords (OTPs)



---

---

---

---

---

---

---

## Example



```
username = input("Enter username: ")
password = input("Enter password: ")
if username == "admin" and password == "1234":
    print("Access granted")
else:
    print("Access denied")
```




---

---

---

---

---

---

---

---

## Authentication



- Make a prediction about what you think will happen, then run the code and note whether your prediction was correct.

```
valid_usernames = ["Rachel", "Louise", "Laura"]
username = input("Enter your username")
while username not in valid_usernames:
    print("Entry denied")
    username = input("Enter your username")
print("Entry granted")
```

- Modify the code so that Emma is now a valid username and adjust the output so that all valid users are greeted with their name e.g. "Emma, entry granted"




---

---

---

---

---

---

---

---

## Summary

- Defensive Design involves:
  - Anticipating misuse and handling errors gracefully.
  - Implementing authentication to protect systems.
- Benefits:
  - Prevents errors.
  - Enhances security.
  - Improves user experience.




---

---

---

---

---

---

---

---

## Check in

Review the following code snippet:

```
password = input("Enter password: ")  
if password == "password":  
    print("Welcome!")
```

Which of the following statements are true?

- A. The password is strong and secure.
- B. The code allows any string input to be used as a password.
- C. The code lacks error handling for empty inputs.
- D. The password can be easily guessed, making the system insecure.



---

---

---

---

---

---

---

## Validation

- Validation checks system inputs to make sure that the data entered is acceptable
- Validation can't check if the data is correct. If you tell it your name is Bob it will assume you're Bob
- Make sure your input statements have clear prompts, that describe what is required and in what format.



---

---

---

---

---

---

---

## Validation Checks

- There are different types of validation, these include:
  - Range check
  - Lookup check
  - Type check
  - Presence check
  - Length check
  - Format check



---

---

---

---

---

---

---

## Range Check

[Trinket Link](https://trinket.io)

- A range check makes sure a number is within a certain range:

```
number = int(input("Please enter a number between 1 and 10"))
while number < 1 or number > 10:
    print("Invalid choice")
    number = int(input("Please enter a number between 1 and 10"))
print("Valid choice")
```



---

---

---

---

---

---

---

## Lookup Check

[Trinket Link](https://trinket.io)

- A lookup check only allows a limited list of items to be entered:

```
choice = "Yes"
while choice in ["yes", "Yes", 'YES', "YES", 'Y', 'y']:
    print('Hello World')
    choice = input("Would you like to see the message again?")
print("OK bye")
```



---

---

---

---

---

---

---

## Type Check

[Trinket Link](https://trinket.io)

- A type check ensures that the correct type of data is entered:

```
number = input("Enter a number")
while not number.isdigit():
    print(number, "is not a number")
    number = input("Enter a number")
print(int(number), "is a number")
```



---

---

---

---

---

---

---

## Presence Check:



- A **presence check** ensures that a required field is not left blank.
- This is crucial to ensure the program has all the necessary data to function correctly.

```
name = input("Enter your name: ")

# Check if the input is empty or only contains spaces
while not name.strip():
    print("This field cannot be left blank.")
    name = input("Enter your name: ")

print("Hello", name)
```




---

---

---

---

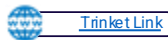
---

---

---

---

## Length Check



- A **length check** ensures that a string is the correct length
- The `len()` function determines the length of the string

```
valid = False
while not valid:
    password = input("Please enter your password: ")
    if len(password) < 8:
        print("Password must have at least 8 characters")
    else:
        print("Password is the correct length")
        valid = True
```




---

---

---

---

---

---

---

---

## Format Check



- A **format check** ensures that input data matches a specific format or pattern.
- For example, validating a password or email address

```
valid = False
while not valid:
    password = input("Please enter at least one number: ")
    for i in ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"]:
        if i in password:
            valid = True
    if not valid:
        print('Invalid password')

print('Valid password')
```




---

---

---

---

---

---

---

---



## Check in

Charlie is developing an adding game with the following rules:

- The player is asked 3 addition questions.
- Each question asks the player to add together two random whole numbers between 1 and 10 inclusive.
- If the player gets the correct answer, 1 is added to their score.
- At the end of the game, their score is displayed.




---

---

---

---

---

---

---

---

## Check in

**Which type of validation would be most suitable for checking the player's answer in Charlie's game?**

- A range check ensures the player's answer is within 1 and 10.
- A length check ensures the player enters exactly 3 numbers.
- A presence check ensures the player submits an answer.
- A type check ensures the player enters a valid integer as their answer.

Post your answers in the chat, e.g. A, B, C, or D.




---

---

---

---

---

---

---

---

## Maintainable Code

- Use subroutines to simplify code.
- Apply consistent naming conventions.
- Proper indentation for readability.
- Add meaningful comments for clarity




---

---

---

---

---

---

---

---

## Subprograms



- Subprograms (functions & procedures) help break the code into smaller, reusable pieces.
- They make the program easier to debug and maintain.

```
def validate_age_input():
    user_input = input("Enter your age: ")
    if user_input.isdigit():
        print("Valid input")
    else:
        print("Invalid input")
```

```
# Calling the subroutine
validate_age_input()
```



## Naming Conventions

- Use clear, descriptive names for variables and subroutines.
- Follow consistent naming styles (e.g. snake\_case for variables).

```
def validate_age_input():
    user_input = input("Enter your age: ")
    if user_input.isdigit():
        print("Valid input")
    else:
        print("Invalid input")
```



## Proper Indentation

- Indentation makes code easier to read and understand.
- Python requires consistent indentation for correct execution.

```
def validate_age_input():
    user_input = input("Enter your age: ")
    if user_input.isdigit():
        print("Valid input")
    else:
        print("Invalid input")
```

```
# Calling the subroutine
validate_age_input()
```



## Meaningful Comments

- Comments explain what the code does, improving readability.
- Use comments to describe logic, complex sections, or functions.

```
def validate_age_input():
    # Prompt the user to enter their age
    user_input = input("Enter your age: ")
    # Check if the input consists only of digits
    if user_input.isdigit():
        # Input is valid if it contains only digits
        print("Valid input")
    else:
        # Input is invalid if it contains non-digit characters
        print("Invalid input")
```




---

---

---

---

---

---

---

---

## Spot the Errors


[Trinket Link](#)


Activity 2

- **Predict** what you think will happen when the code is run. Does it handle all inputs correctly?

```
def validate_age():
    age = input("Enter your age: ")
    if age > 0 and age < 120:
        print("Valid age")
    else:
        print("Invalid input")
```

- **Run** the code and identify the errors.
- **Modify** the code so that:
  - The input is validated as a positive integer.
  - The program doesn't crash if invalid input is entered.




---

---

---

---

---

---

---

---

## Intended learning outcomes

- Be able to explain the importance of defensive design in creating robust programs.
- Be able to implement input validation techniques to ensure data integrity.
- Be able to write simple authentication routines (e.g. username and password checks).
- Be able to apply techniques for writing maintainable code, including naming conventions, subprograms, and commenting.




---

---

---

---

---

---

---

---

# Game Board

## Robust Programs

1	User input	Programming fundamentals (Programming concepts)	SCORE 0/0/0
2	Converting data types	Programming fundamentals (Programming concepts)	SCORE 0/0/0
3	Accepting input in Python	Programming fundamentals (Programming concepts)	SCORE 0/0/0

Save to My gameboards

---

---

---

---

---

---

---

---

# ISAAC boosters

MY ACCOUNT
 LOGIN
 SEARCH

MY BOARD
 TRACKING
 LABELS
 BOOSTERS
 BOOSTER BOARD

Your email address is not verified - please find our email in your inbox and follow the verification link. You can [manage your email subscription preferences](#). To change your account email, go to [My Account](#).

Snooze

## Welcome

For your students

[GCSE resources](#)
[A Level resources](#)
[Events](#)

For you

Browse our National Centre for Computing Education courses to help you teach computing across the secondary curriculum

[Key stage 3 courses](#)
[Key stage 4 courses](#)
[A level courses](#)

### I belong in Computer Science posters

The I belong poster series celebrates diversity in Computer Science. We believe that whatever your race, religion, ethnicity, gender, sexuality, or neurodiversity, everyone belongs in Computer Science. Teachers working in schools in England can request a free set of I belong in Computer Science posters to display in their classrooms.

[Order your posters](#)

Keep an eye out for more student booster events

---

---

---

---

---

---

---

---

# Thank you

---

---

---

---

---

---

---

---