

# 2 - Design

NEA

# Variables & other Data Structures


What data structures do you intend to use and what names will you use for them?

They could include:

- variables
- arrays (lists)
- text files
- database tables

For inputs you should also state any validation that you intend to use.

Validation could include:

- presence check
  - range check
  - length check
  - type check
  - lookup check
  - format check
- 

# Designing Variables

Name	Data Type	Purpose	Validation
level	Integer	Stores the difficulty level, which is the number of screens the player will see in one game	Type check: must be an integer Range check: must be 3-10

<b>Name</b>	What will the variable be called? This should match any variables referred to in your algorithm design section.
<b>Data Type</b>	<p>The most likely will be:</p> <ul style="list-style-type: none"><li>• Integer (whole numbers only, including zero and negatives)</li><li>• Real (numbers that can be fractions or whole numbers)</li><li>• Character (a single letter, number or symbol)</li><li>• String (a sequence of characters stored as a single item)</li><li>• Boolean (can only be only true or false)</li><li>• Date/time</li></ul>
<b>Purpose</b>	What will be the role of this variable in your program? Why is it needed?
<b>Validation</b>	<p>What checks will be in place to ensure valid data is entered? You might include more than one for a single piece of data:</p> <ul style="list-style-type: none"><li>• Presence check (makes sure something has been entered)</li><li>• Type check (makes sure the data type is adhered to)</li><li>• Range check (makes sure a number or date is below a certain value, above a certain value or between two values)</li><li>• Length check (makes sure the correct number of characters has been entered)</li><li>• Lookup check (checks that the data entered matches an item on a list)</li><li>• Format check (makes sure types of character are in the right place, such as in a date, postcode or National Insurance number)</li></ul> <p>It is possible that some data items will not need a validation check, especially if the user does not interact with them directly. In this case, simply write 'none'.</p>

# Designing arrays

Designing an array is similar to designing a variable, with the additional field 'size'.  
This should say how many elements will be stored within the array.

Name	Data Type	Purpose	Validation	Size
numbers	Integer	Stores the user's answer, with one digit in each element	Type check: must be an integer Range check: must be 0-9	10

# Designing files

Name	Type	Description
results	.txt	<p>When a user has completed the game, their score will be saved, along with their name and the data that they played. Each piece of information will be separated by a hash sign, e.g.</p> <p>8/10#Bob Jones#31/05/2015</p> <p>There is no validation for data entering this file, as all data would already have been validated when user input was stored in variables.</p>

<b>Name</b>	The name of the file
<b>Type</b>	The file extension of the file. In this case, as in many cases, it is a text file, which has the extension .txt
<b>Description</b>	<p>Files can consist of any number of formats and rules, so it's much better to have a 'description' column in this table than lots of smaller columns. Here, you might describe:</p> <ul style="list-style-type: none"><li>• the purpose of the file</li><li>• exactly what data is stored here, and when</li><li>• an example piece of data</li><li>• any validation information</li><li>• anything else that would be needed in order for someone to then create this file</li></ul>

# Designing Records

This design would be implemented within a database program. To clarify, this is the design for **one** record, not five. Each record here will consist of five fields and each field would be designed as part of that record.

Don't worry if your program does not consist of all four of variables, arrays, files and records; programs are not expected to. They are all illustrated here because different programs might use different combinations of them.

Name	Data Type	Purpose	Validation	Example
ID	Integer	Primary key - to identify each game uniquely	None - automatically generated	1
GameDate	Date	Identifies when the game was played	Range check: must be before tomorrow	31/05/2015
Level	Integer	The maximum score of this game	Type check: must be an integer Range check: must be 3-10	5
Score	Integer	The actual score of this game	Type check: must be an integer Range check: must be between 3 and the value of 'Level'	4
User	String	The name of the person who played the game	Presence check: must contain something	Bob Jones

# Algorithms – Pseudocode

You now need to write Pseudocode for some of your design

```
START
    level = (see 'check the level' process)
    new integer (count) = 0
    new integer (random)
    new integer array (numbers) [level]
    WHILE count < level
        random = new random number (0 - 9)
        display random
        numbers [count] = random
        wait 1 second
        count = count + 1
    END WHILE
    new string (answer) = (user input)
END
```

Pseudocode is more open to variation than flowcharts. Good pseudocode can generally be described as being:

- Somewhere between English and any programming language
- Consistent – however you declare one variable, declare all others in the same way
- Rigid in terms of code structures – just like the WHILE / END WHILE structure is clearly defined with familiar keywords and good indentation, IF / END IF structures would be shown similarly.

# Algorithms – Pseudocode

As you write your Pseudocode refer back to your Test Plan and Success Criteria to make sure your solution is appropriate.





# Design Checklist

## Design

<input type="checkbox"/> The algorithms are either incomplete or missing	<input type="checkbox"/> The algorithms are complete, but not fully functional	<input type="checkbox"/> The algorithms are fully complete and clearly functional
<input type="checkbox"/> The intended approach is not discussed or justified	<input type="checkbox"/> The intended approach is discussed but not justified	<input type="checkbox"/> The intended approach is both discussed and fully justified
<input type="checkbox"/> The user interface is not planned, or planned but not functional	<input type="checkbox"/> The user interface is planned but may not be fully functional	<input type="checkbox"/> The user interface is fully planned and functional
<input type="checkbox"/> The design could not be used to construct the intended solution	<input type="checkbox"/> The design could be used to construct the intended solution with some guidance	<input type="checkbox"/> The design could be used to construct the intended solution without guidance
<input type="checkbox"/> The test plan covers some basic functionality but is limited	<input type="checkbox"/> The test plan covers most functionality with some robustness testing	<input type="checkbox"/> The test plan covers full functionality and covers robustness testing
<input type="checkbox"/> There is limited discussion of variables and data structures	<input type="checkbox"/> There is discussion of most variables and data structures	<input type="checkbox"/> There is justification of all data structures and variables
<input type="checkbox"/> There is little evidence of modular designs	<input type="checkbox"/> Most of the program is designed in a modular way – but not effectively	<input type="checkbox"/> An effective modular design is produced