# Integrated Learning Environment

## Generic

### Expression evaluator pane

Permits only expressions to be entered – anything else is rejected.

Scrolling window, but cannot edit anything other than current entry (though this can wrap onto multiple lines). However, can get back previous entries via cursor up.

Cannot copy and paste

Clear button clears screen and all memory of previous entries

(Optional) If the return type matches a bitmap pattern, this will be displayed on the graphic screen instead.

Need to include a typeof or equivalent for use in evaluator

All instances should be able to render themselves in a friendly way. If necessary can define extension methods or functions for any types where the ToString (or equivalent) is not friendly enough.

### Available functions

Potentially includes standard functions for the language, and also common MetalUp library – which includes a Random generation capability.

Can view as a list, or search by name. Brings up a small help page, showing signature and explanation.

There will be behind-the-scenes Import/Using statements to permit access to these functions without using namespaces.

*No function/method may be called that is not on the list, even if it is in the same namespace.* (This will be enforced by simple text-searching – this might throw some false positives but that’s OK, the message will say e.g. ‘Function Foo may not be used’).

By this means, no I/O functions may be used in expressions (or in function definitions). Also, we can disallow other non-pure functions such as list mutating methods.

### Function definition pane

Can only reference the available list of functions & data definitions, plus other UDFs in the pane.

Cannot copy and paste into the function definition pane.

Configuration option can restrict function bodies to a single return expression.

### Feedback pane

Compile and runtime messages – to keep separate from expression evaluator.

## Assignment

An Assignment has all the features of the Generic REPL, but with various optional additions.

### Task

Description of task and what to do

### Tests

### Available functions and data definitions

The list of pre-defined functions available may be specific to each task.

In addition, there may be data definitions (values and structures) to be used for manual testing (via expression evaluator) or by the functions themselves. Where language permits, these should be defined as constants or immutable reference types.

### Marking

Each task is given a maximum mark, which will be reduced progressively by the use of hints.

When used in freestanding mode, a ‘certificate of completion’ will be displayed, with a prompt to save or print it.

The cert will have a number hashed from the user’s name and task details that may be verified.

In a future system marks may be recorded automatically on the management system instead of

### Hints

Multiple levels of hints, costing progressively more marks as used.

Must be used in order. Each hint (after first) builds explicitly on the previous one, so user can see what they should have been able to do with that hint.

May be an enforced delay before next hint becomes available.

Example hierarchy:

* Hint two describes algorithm in words
* Hint one, specified which functions/data you need to use.
* Hint three is partial code with fill-in-the-blanks
* Hint four is complete code, but shown as image that cannot be cut/pasted – must still be copied into function definition area

### Context code

There may be invisible context code that can be run only when the function is complete and correct – for example to show a game or other application being built up. (Would be nice to allow the function code to be deliberately modifiable to show that the app really is using it).

Any call to UDF should trap all exceptions thrown by the function and present clearly.

Could have option to permit some or all of this context code to be viewed (but not edited).

### Task creation

There should be a standalone mechanism (e.g. a language-specific template to be run within an IDE) whereby an author can create a task, and then combine the required elements to create a standalone task page.

Such pages could then be added to the main website, each with a unique page ‘name’ that cannot reliably be guessed at.

### Configuration options

* Whether user may write more than one function to complete task or not
* Whether function(s) must be strictly in the form ‘return [expression]’ or may permit multiple statements.

## Management system

Manages users, tasks, results.

Records timings of log on, each code submission (with code and result) and each request for hint. The teacher can e.g. see how long a user spent on the task, in total or before each hint was asked for, and render this visually.

### Anti-plagiarism

Prevent pasting in of text into the coding box on all tests.

System can look for identical wrong code being pasted and/or correlation of results /activity between pupils, or inconsistency between class and prep behaviour/results. Can show this visually on a chart of when done

## Extended development projects

Projects can be an application (e.g. a game) or just a set of related pieces of work (e.g. statistics)

Need not be completed contiguously. Pupils might work on several over a term, coming back to them when the requisite skill has been taught.