

# Achievement Standard Digital Technologies 91637

Develop a complex computer program for a specified task

Version 3

Level 3

Credits 6

Achievement	Achievement with Merit	Achievement with Excellence
Develop a complex computer program for a specified task involve.	Skilfully develop a complex computer program for a specified task involves.	Efficiently develop a complex computer program for a specified task involves.

## Student Instructions

### Introduction

This assessment involves creating a complex computer program for a specified task. Your program must include a graphical user interface, class(es), objects and at least one indexed structure (such as a list). Please choose from the options below:

#### Option 1

In consultation with your teacher, develop a computer program to solve a real life problem of your choosing. Note that time is limited for completing the planning, creation and testing of your program so be careful not to be overly ambitious in the design. The program you create should be more than simply re-purposing code that you have previously developed for a practice task. Note that you **must** get teacher approval for the program that you wish to develop.

#### Option 2

Write a program based on a game of chance. To begin, players should 'pay' a given amount. They should then be able to play the game until they decide to quit / they have lost all their money. In addition to clearly displaying the amount won / lost, the game should allow users to access a help / information screen. An idea for a suitable game is provided in the Program Details section below. This is a guide only and you are welcome to change the odds, rules of the game in consultation with your teacher.

This is an individual assessment activity. You will have four weeks of in-class and out-of-class time to complete the planning of the program, and to complete the programming component.

See Appendix 1 for planning guide.

## Game of Chance Program Details

- In our example game “Lucky Unicorns”, users are given three mystery boxes. The boxes can contain a heart, a star, a jack-in-the-box or a unicorn. It costs \$1.00 to buy three boxes. Users then open the boxes (by clicking on them) and the program should tell them how much they have won (if anything). Below is a chart showing possible winning combinations.

	Pay-out
3 Unicorns	\$5.00
2 Unicorns	\$2.50
3 ♥'s	\$2.50
3 's	\$2.50
3 Jacks	\$1.00
1 Unicorn, 2 others	\$1.50
2 ♥'s / 2 's	\$1.00

- You are welcome to change the number of boxes, the pay-out amounts and items if desired. When designing the game you should ensure that the ‘house’ has an advantage (ie: over time users should lose more than they win). You can use the ‘unicorn pay-out calculator’ to change winning combinations / the pay-out amount for a game with three items and four boxes. You should ensure that your code is flexible, robust and easy to edit (ie: if your pay-out does not give the house a sufficient advantage, it should be easy to adjust the pay-out amounts etc)
- Your design should allow your program to use at least one indexed structure (eg: a list).
- The interface should allow the user to put in an amount of money to begin playing. It should also have a ‘help / instructions’ button which links to a screen displaying the rules of the game.
- Your game should limit the amount of money that users can put in at the start of the game so that players do not risk losing excessive amounts of money.
- There should be a way for the user to initiate the boxes being opened. The program should then inform the player of the outcome and keep track of the amount of money won / lost. The player should be able to quit at any time (and take their winnings). If the player has no money left, then they should not be allowed to initiate a further round.
- The program should display relevant error messages for appropriate situations.

## Final Submission

At the end of the allotted time, hand in the following for assessment:

- your planning document (complete the tasks in appendix 1 below to help generate this)
- a pdf print-out of your final source code
- an electronic version of your source code
- screenshots of example output with annotations

## Assessment schedule

	Required Elements	Evidence
<b>Achieved</b>	Designing and implementing a program that includes variables, an indexed data structure (a list), and a modular structure including details of the procedural structures of the modules.	Has a working program.
	Including a working graphical user interface with different sources of event generating components and event handling.	Program has a functional graphical user interface.
	Using class(es) and objects to encapsulate data and methods.	Evidence in program code.
	Setting out the program code clearly and documenting the program with comments.	Evidence in program code.
	Testing and debugging the program to ensure it works on a sample of expected input cases.	Evidence in program code and planning.

	Required Elements	Evidence
<b>Merit</b>	Using well-chosen modular and procedural structures, scope and encapsulation for data and methods, graphical user interface and event handling mechanisms.	Evidence in program code.
	Documenting the program with variable and module names and comments that accurately describe code function and behaviour.	Evidence in program code.
	Following a disciplined design and implementation process.	Evidence in planning and program.
	Develops with documented cycles of incremental development.	Evidence in planning and program.
	Implements comprehensive testing process, to ensure that the program works on inputs that include both expected and boundary cases.	Evidence in planning document.

	Required Elements	Evidence
<b>Excellence</b>	Ensuring that the overall modular and procedural design, graphical user interface, and event handling design, are a well-structured, logical decomposition of the task, and that the program is flexible and robust.	Evidence in planning and program.
	Setting out the program code concisely and documenting the program with comments that explain and justify decisions.	Evidence in planning and program.
	Comprehensively testing and debugging the program in an organised and time effective way to ensure the program is correct on expected, boundary and invalid input cases.	Evidence in planning and program.

## Appendix 1: Planning Guide

### Task 1: Identify user inputs

*What program functions can the user trigger through the interface?*

### Task 2: Identify information to be displayed

*What information will the interface need to display to the user?*

### Task 3: Sketch interface design

*Draft a rough design for the interface that allows the user to trigger functionality in task 1, while also annotating where the information in task 2 will be displayed. Create another sketch listing the interface widgets used to create the interface.*

### Task 4: Identify any class(es) required

*Explain what the class will represent, plus listing what information will be stored in the class and any functions the class will have.*

### Task 5: Identify any constants or existing data if required

### Task 6: Identify indexed data structures

### Task 7: Determine what calculations are necessary

*Write out the calculations the program will have to compute.*

### Task 8: Develop a modular structure for your program

*Describe any functions that the computer program will have, identifying any sub-functions where required.*

### Task 9: Define the functions identified

*Describe the functions for both the main program and any classes in terms of input and/or output where required. You may choose to do this with flow charts or pseudo-code (not Python code!). Add in additional steps or explanations using sequential, conditional, iterative statements where required. Identify global and/or local variables.*

### Task 10: Document test cases for testing the program

*Document any testing that can be used to test your program. If any input is inputted using the keyboard, describe the expected input, plus any exceptional, boundary or invalid cases.*

### Task 11: Refine the plan

*Note any modifications here when iterating through the development cycles.*

### Task 12: Document testing

*Show screenshots of your program working with descriptions of each image. These images should test the tests cases listed above.*