Level 6 – New Zealand Curriculum

DIGITAL TECHNOLOGIES & HANGARAU MATIHIKO

Teaching and learning programme

Planning & Programming (Python)

MINISTRY OF EDUCATION
TE TĀHUHU O TE MĀTAURANGA

Developed by Jennifer Gottschalk, Whangaparaoa College, 2018

Published 2018 by the Ministry of Education
PO Box 1666, Wellington 6011, New Zealand

www.education.govt.nz
All rights reserved
Copyright © Crown 2018

Summary of the teaching and learning programme

This programme is based around an eBook tutorial which includes embedded video. Students are given several problems related to games of chance and are encouraged to develop programs to solve the problems. No prior programming knowledge is assumed.

By the end of this teaching and learning programme, students will be able to:

- decompose a problem into smaller sub-problems
- create and test code that solves each problem
- combine the code into a fully functional program
- test that the program is easy to use.

Duration

Teachers should adjust this programme to suit their students and timetables. One credit equates to 10 notional hours of teaching, practice and/or study, and assessment.

Key teaching and learning concepts – the big ideas

- A problem can be decomposed into smaller sub-problems
- Each sub-problem can be developed and tested
- The code fragments can then be composed into a fully functional program
- Testing should include boundary and unexpected values
- Functions can be used to avoid repeating code
- Usability is important! Volunteers can be asked to test code and changes can then be made to ensure that the final outcome is easy to use.

Alignment to NZC and/or Te Marautanga – (DTHM progress outcomes and progressions)

This material is focused on the *designing and developing digital outcomes* progression. Students will:

- design and develop a basic program
- ensure that their outcomes are easy to use (preferably by including some usability testing as part of the process)
- be ethical when it comes to designing and creating their outcome (specifically they will honour copyright.)
- meet the criteria set out in achievement standards 91883 and 91884.

Links to other learning areas

This programme involves documenting three books that students have read and could be used in conjunction with English achievement standard 90854.

Teaching and learning pedagogy

The programme uses ‘flipped’ learning, where the process has been videoed and students are encouraged to create their own programs by following the video tutorials. They are also encouraged to go beyond the basics where possible. By using an eBook with video, teachers are free to work with individuals and troubleshoot in a way that would not be possible using more traditional methods. Teachers could encourage students to collaborate and work in small groups during the learning phase for this standard.

Prior knowledge and place in learning journey

No prior knowledge is assumed. If students have programmed before, this programme will help formalise their learning and also ensure that they develop good coding (and design) habits.

Resources required

- The learning resource for this programme found in the ePub and support files. Note that this includes 'teacher only' answers and a Teaching Guide. All of the resources are found here.
- Python
- Google Documents / Word

Software used:

- Python 3.x
- Pycharm (optional but very useful).

How you might adapt this in your classroom

The tasks in the support files can be modified to suit a given class. Each task has suggestions for extension activities, and students should be encouraged to go beyond the basics if they have prior programming experience. While the resource is focused on Python programs, students could be encouraged to develop similar outcomes in the language of their choice.

Assessment

There is an assessment task included at the back of this programme. The default task asks students to create a mathematics quiz.

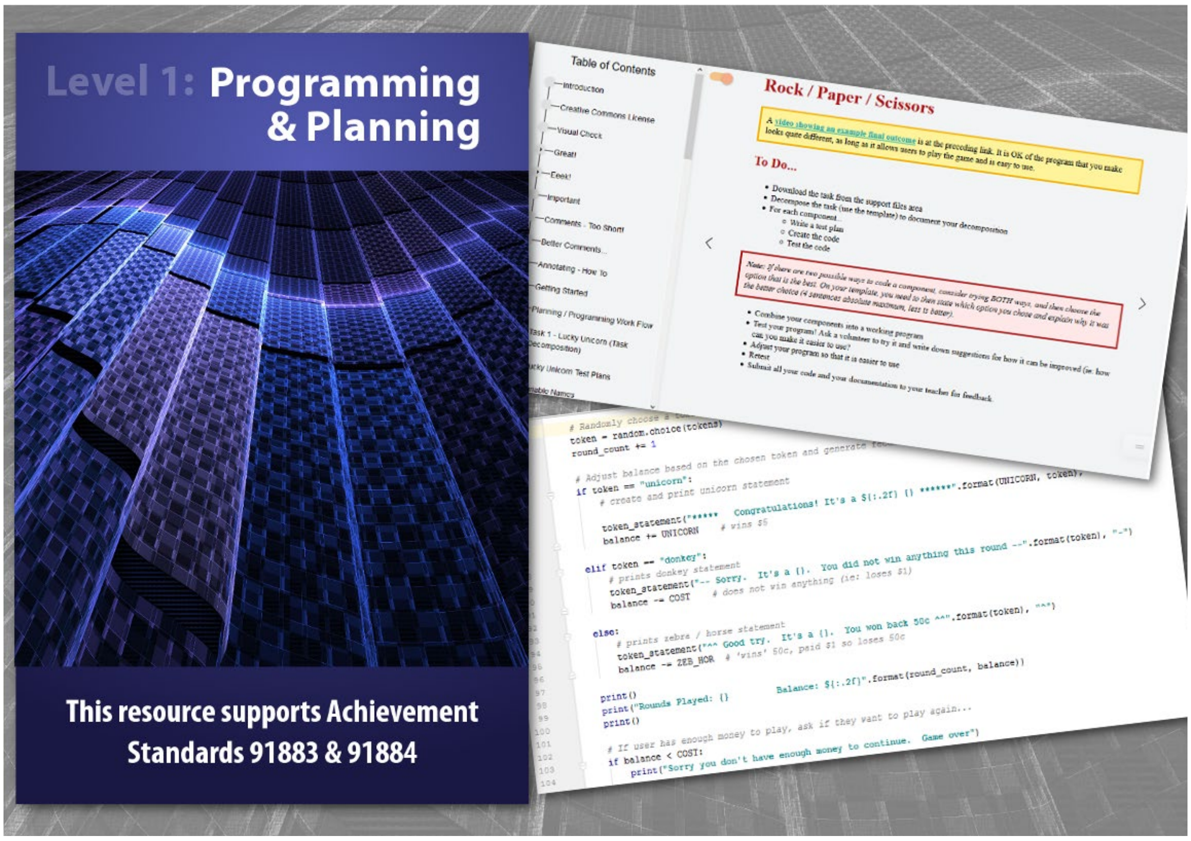
Level 1: Programming & Planning
This resource supports Achievement Standards 91883 & 91884

Table of Contents

Rock / Paper / Scissors

A video showing an example final outcome is at the preceding link. It is OK of the program that you make looks quite different, as long as it allows users to play the game and is easy to use.

To Do...

- Download the task from the support files area
- Decompose the task (use the template) to document your decomposition
- For each component...
  - Write a test plan
  - Create the code
  - Test the code

Note: If there are two possible ways to code a component, consider trying BOTH ways, and then choose the option that is the best. On your template, you need to then state which option you chose and explain why it was the better choice (4 sentences absolute maximum, less is better).

- Combine your components into a working program
- Test your program! Ask a volunteer to try it and write down suggestions for how it can be improved (ie: how can you make it easier to use?
- Adjust your program so that it is easier to use
- Retest
- Submit all your code and your documentation to your teacher for feedback

```
# Randomly choose a to[illegible]
token = random.choice(tokens)
round_count += 1

# Adjust balance based on the chosen token and generate fe[illegible]
if token == "unicorn":
    # create and print unicorn statement
    token_statement("*****   Congratulations! It's a ${:.2f} {} ******".format(UNICORN, token),
    balance += UNICORN    # wins $5

elif token == "donkey":
    # prints donkey statement
    token_statement("-- Sorry.  It's a {}.  You did not win anything this round --".format(token), "-")
    balance -= COST    # does not win anything (ie: loses $1)

else:
    # prints zebra / horse statement
    token_statement("^^ Good try.  It's a {}.  You won back 50c ^^".format(token), "^")
    balance -= ZEB_HOR  # 'wins' 50c, paid $1 so loses 50c

print()
print("Rounds Played: {}            Balance: ${:.2f}".format(round_count, balance))
print()

# If user has enough money to play, ask if they want to play again...
if balance < COST:
    print("Sorry you don't have enough money to continue.  Game over")
```

TERM OUTLINE

Week 1	Weeks 2 & 3	Weeks 4 & 5	Weeks 6 & 7	Weeks 8 & 9	Weeks 10 & 11	Weeks 12 & 13
The basics	Lucky unicorn game	Higher/lower game	Rock, paper, scissors	Collect them all	Car racer game	Assessment

Resources – The material below is used for each task:

- ePub
- Support files
- Documentation template
- Program planning helper

Teaching and learning programme

What is being covered	Approximate duration	Specific learning outcomes *Students will be able to:*	Learning activities	Checkpoints
The basics	5 hours maximum	• name variables • get user input • loop code • cope with unexpected input • write functions.	• Discuss the importance of commenting code • Learn how to name variables correctly • Learn how to get user input and the difference between strings and integers • Learn how to write 'if' statements and use <, >, ==, etc. • Learn how to write 'while' loops • Learn how to implement try or except code • Combine what we have learned to create a number checker • Change our number checking code into a function	None

What is being covered	Approximate duration	Specific learning outcomes *Students will be able to:*	Learning activities	Checkpoints
Lucky unicorn game	10 hours	• break down the task • code and test each section • create a fully working program from the components • test that the program is usable.	• Analyse the 'Lucky unicorn' problem and break it down into a series of smaller components • Create test plans for each component (note that this can be done at the start of the task or test plans can be generated before creating a given component) • Learn how to create 'for' loops (ie, 'Loop Interlude') • Randomly choose items from a list • Check that the probability of getting a unicorn is not too high • Set up payment mechanics • Set up end-game mechanics • Combine all the sub-programs into a fully functioning program • Test the usability of the program • Fix the output statements so they are easy to read • Retest that the program works correctly.	**Checkpoint 1:** Submit the entire 'Lucky unicorn' program and the associated documentation for feedback. The program may well be different to the one in the video walk-through.
Higher/lower game	10 hours	• break down the task • code and test each section • create a fully working program from the components • test that the program is usable.	• Decompose the problem and create test plans for the various components (note that test plans can be created at a later date if preferred, as long as they are generated before the code for that section is written) • Create the game by using what you learned in 'Lucky unicorn' and following the provided videos for 'new' code • If possible, create a fully featured game rather than a basic version.	**Checkpoint 2:** Submit the entire 'Higher/lower' program and the associated documentation for feedback. The program may well be different from the one in the video walk-through.

What is being covered	**Approximate duration**	**Specific learning outcomes** *Students will be able to:*	**Learning activities**	**Checkpoints**
Rock, paper, scissors	10 hours maximum	• break down the task • code and test each section • create a fully working program from the components • test that the program is usable.	• Decompose the problem and create test plans for the various components (note that test plans can be created at a later date if preferred, as long as they are generated **before** the code for that section is written) • Create the game by using what you have learned in previous tasks • If possible, make the game your own. Consider how you'd like the scoring to work (eg: best of x or the first to x). **Teacher note:** *Students should be able to compare user and computer choice using* ***five*** *'if' statements. If they want to use more than that, encourage them to think about how they can do it in less. Using more statements is OK and will work but it is inefficient and probably indicates that the student is working at an A level.*	**Checkpoint 3:** Submit the entire 'Rock, paper, scissors' program and the associated documentation for feedback.
Collect them all	10 hours maximum	• break down the task • code and test each section • create a fully working program from the components • test that the program is usable.	• Decompose the problem and create test plans for the various components (note that test plans can be created at a later date if preferred, as long as they are generated **before** the code for that section is written) • Create the tool by using what you learned in previous tasks. It is OK to recycle or repurpose functions that you have used in previous tasks.	**Checkpoint 4:** Submit 'Collect them all' and the associated documentation for feedback.

What is being covered	**Approximate duration**	**Specific learning outcomes** *Students will be able to:*	**Learning activities**	**Checkpoints**
Car racer game	10 hours maximum	• break down the task • code and test each section • create a fully working program from the components • test that the program is usable.	• Decompose the problem and create test plans for the various components (note that test plans can be created at a later date if preferred, as long as they are generated **before** the code for that section is written) • Create the game by using what you learned in previous tasks. It is OK to recycle or repurpose functions that you have used in previous tasks. **Teacher Note:** *There are a wide number of possible correct solutions to this task. Students could be encouraged to use the turtle module and race turtles instead of cars. This entire exercise is an extension task, and if students don't get this far, that is not a problem. This is really an opportunity for students to explore.* ***The solution uses 2-dimensional arrays and this is not required at level 1. Very top students might need to know that functions can only return one thing but that thing can be a list.***	**Checkpoint 5:** Submit your 'Car racer' game and the associated documentation for feedback**.**
Assessment activity	10 hours	The recommendation is to give students two weeks of class time.		

ASSESSMENT TASK

Achievement standard:	91883 and 91884
Standard title:	Develop a computer program (4 credits) Use basic iterative processes to develop a digital outcome (6 credits)
Total credits:	10

OVERVIEW

You are going to develop a basic quiz program that can be used with a chosen audience.

This assessment activity requires you to plan, trial, test and develop the quiz program. You will need to manage the development by decomposing the program into smaller components. You will also need to trial each component and develop your program in an iterative manner. It is important to test your program and provide evidence that it functions as intended. Finally you need to describe and address the relevant implications.

Please be aware that NZQA have read the assessment task but it will still need to be checked by the teacher using the assessment to ensure it meets all requirements.

HOW WILL YOU BE ASSESSED?

You will be assessed on how effectively you plan your development, decompose the problem into smaller components, trial these components, and test and refine your program so that it is a high-quality response to the task (eg, well-structured, logical, flexible, robust and comprehensively tested).

When planning and developing your program, you must ensure that it uses:

- variables storing at least two types of data (e.g. numeric, text, Boolean)
- sequence, selection and iteration control structures
- input from a user, sensors or another external source

and one or more of:

- data stored in collections (e.g. lists, arrays, dictionaries)
- user-defined methods, functions or procedures.

You must use conventions for the programming language and code commenting that describes code function and behaviour. Your program should be comprehensively tested and debugged in an organised manner.

TASK

Your quiz can be on the topic of your choice provided that the questions you ask do not break copyright and are suitable for your chosen audience (ie, not too easy and not too hard).

You can make any of the following:

- a maths quiz (recommended)
- a word answer quiz
- a game format quiz
- a multiple choice quiz
- a multi-player quiz
- any other style of quiz.

ASSESSMENT TASK

Achievement standard:	91883 and 91884
Standard title:	Develop a computer program (4 credits) Use basic iterative processes to develop a digital outcome (6 credits)
Total credits:	10

TASK

You need to think about:

- Who your quiz is aimed at
- What user inputs you will need
- How the questions and answers will be stored
- How you will give feedback to the user
- Describing and addressing the relevant implications.

Planning:

You must:

- Plan how you will develop your quiz by decomposing the problem into smaller or sub-components
- Create a test plan that allows you to confirm that your program works for expected and relevant boundary cases.

Developing and testing:

You must:

- Try various options when creating your quiz. You should choose the options which work best and provide evidence of your trialling including the iterative trialling of components.
- Comment your code.
- Code, test and debug each component or sub-component before moving on. Create a new version of your program for each iteration.
- Provide evidence that you have iteratively trialled the program components.
- Systematically test your code (follow your test plan). Include screenshots (or video evidence) showing that your program's output matches the plan's expected values and has been tested with relevant boundary cases.

Additional evidence

Provide evidence to show how you have described and addressed the relevant implications for your program, for example:

- How your quiz will be suitable for your chosen audience
- How you will honour copyright
- How you will make sure that your quiz has taken into account relevant HCI principles.

HAND IN

- Your plan and testing evidence
- All the code that you have developed including the code for each component. Please clearly name your files so that it is easy to identify the code for the final, working outcome.

TEACHER GUIDELINES

The following guidelines are supplied to support teachers/kaiako to carry out valid and consistent assessment using this internal assessment resource.

Teachers/kaiako need to be very familiar with the outcome being assessed by the achievement standard/s. The achievement criteria and the explanatory notes contain information, definitions and requirements that are crucial when interpreting the standard and assessing students/ākonga against it.

Please be aware that NZQA have read the assessment task but it will still need to be checked by the teacher using the assessment to ensure it meets all requirements.

CONTEXT/TE HOROPAKI

This assessment activity requires students to develop a computer program that is suitable for the intended audience. Students should use basic iterative processes to develop a computer program.

During this process, students will develop a refined computer program. They need to ensure that they:

- use appropriate tools and techniques
- test the outcome thoroughly
- describe and address any implications and end-user considerations.

It is strongly recommended that students make a maths-type quiz as that will easily allow them to randomly generate unique questions each time the quiz is played. It also allows them to generate material that does not break copyright.

CONDITIONS/NGĀ TIKANGA

Where a group approach is used, the teacher needs to ensure that there is opportunity for each student to provide evidence for all aspects of the standards.

Schedule regular progress checks with the students during this activity.

Conditions of Assessment related to this achievement standard can be found at http://ncea.tki.org.nz

RESOURCE REQUIREMENTS/NGĀ RAUEMI

Students will need access to computers running Python and word processing software to allow them to complete the required documentation.

ADDITIONAL INFORMATION/HE KŌRERO ATU

Planning:

Students should plan how they will develop their computer program.

The planning should include research into issues regarding their chosen theme (eg, writing questions with suitable content and/or ensuring the questions are at an appropriate level for the intended audience). Their planning should also include evidence of breaking down the project into components that need to be included in their outcome. For example, when developing a computer program for the quiz, the components may include generating questions, getting (and validating) user answers, providing feedback on user answers and keeping track of the number of questions that have been answered correctly. This could involve using programming techniques such as, the use of exceptions where required, loops to replace repeated code, functions, variables, collections and various user interface elements.

Planning should be undertaken in a manner that suits the outcome and could include sketches, wireframes, storyboards, or mock-ups. Students may use online interactive or collaborative planning tools.

Trialling:

Students should iteratively trial the components to be included and refine their outcome based on the evidence of their trialling. Students should also provide evidence that they have planned for expected cases to test the outcome and carried out testing to improve and refine their outcome.
The final outcome should include evidence that the student has described and addressed a range of relevant implications and end-user considerations.

Outcome:

Students will produce an individual computer program that is appropriate for the teaching and learning programme. Teachers should ensure the rigour of the outcome is appropriate for level 6 of the NZ Curriculum (eg, has not been produced through simple modification of pre-designed templates and/or drag and drop WYSIWYG applications). The computer program that the student is being assessed on should be original coding, which has been developed by the student. However, they may also incorporate other open-source or royalty-free media that they have not developed, as appropriate to the outcome. For example, they may include an open-source or royalty-free soundtrack or images that they have not developed but have the permission (or appropriate licence type) to include in the outcome.

ASSESSMENT SCHEDULE: DEVELOP A COMPUTER PROGRAM (91883)

EVIDENCE/JUDGMENTS FOR ACHIEVEMENT/PAETAE	EVIDENCE/JUDGMENTS FOR ACHIEVEMENT WITH MERIT/KAIAKA	EVIDENCE/JUDGMENTS FOR ACHIEVEMENT WITH EXCELLENCE/KAIRANGI
Develop a computer program.	Develop an informed computer program.	Develop a refined computer program.
The student has: • written a simple, functional quiz program in the language of their choice. The program may not be structured very well. ***For example (partial evidence)*** The student has written a program that performs the specified task, using a suitable programming language. The program includes: • variables which store at least two different data types • sequence, selection and iteration control structures • input either from a user or an external source • either data stored in lists or arrays or dictionary or user defined methods, procedures or functions. Note that BOTH functions AND lists are not required to pass this standard. The quiz works as expected.	**The student has:** • documented the program with variable names and comments that describe code function and behaviour ***For example (partial evidence)*** There are frequent clear comments throughout the code that helps to describe relevant functions or sections of code. The variable names clearly describe the data they hold: • eg, # this function tests that the user input is a number. It will continue to ask for input until the input is a number. • eg, answer = is_a_number(question)	**The student has:** • ensured that the program is a well-structured, logical solution to the task ***For example (partial evidence)*** The code is clean, concise, efficient and easily readable. The main program may be short and might consist of multiple reusable user defined functions which do most of the logic and processing. The layout might include sections as follows (some of the material might be in a loop to allow multiple questions to be asked until a condition is met) • import modules (eg, import random) • user-defined functions • set-up constants • initialise variables • generate/ask question • calculate correct answer • get user answer and check it is valid • compare user answer with correct answer and update score/counters • give user feedback.

ASSESSMENT SCHEDULE: DEVELOP A COMPUTER PROGRAM (91883)

The student has:

- set out the program code clearly, documenting the program with comments

For example (partial evidence)

Comments are present but may not be particularly descriptive or frequent:

- eg # this code creates the quiz loop

The student has:

- tested and debugged the program to ensure that it works on a sample of expected cases.

For example (partial evidence)

The student has shown some evidence of expected cases that were used to test and debug the program and show that the program works when the user inputs data that is expected.

Testing may be trial and error rather than clearly thought out.

The student has:

- followed conventions of the chosen programming language

For example (partial evidence)

The student has followed common programming conventions for their chosen language.

Python files and functions contain a docstring explaining the purpose of the program or function. For example, in Python

- Constants are ...

- Variable names are ...

Function definitions appear before loose lines of code and the main section of code is all at the bottom, not between the functions, thus making the program easier to read.

The student has:

- made the program flexible and robust

For example (partial evidence)

The student has used methods, functions, procedures, actions and control structures effectively.

User input is checked to ensure that it is valid.

Expected, boundary and invalid user input is handled correctly <see testing>.

Constants, variables and derived values are used instead of hard coded values.

The student uses a series of 'if', 'elif', 'else' statements rather than multiple, single 'if' statements: For example, to check a user answer where 'x' is a special exit code, their code might say:

```
if user_ans == 'x':
    break out of the loop
elif user_ans == correct
    do something
else
    do something
rather than separate statements
if user_ans == 'x':
    break out of loop
if user_ans == correct
    do something
if user_ans != correct
    do something
```

	The student has: • tested and debugged the program in an **organised** way to ensure that it works for expected and relevant boundary cases. *For example (partial evidence)* The student tests frequently during development (observed), and the final program works when the user inputs the data that is expected and checks or handles when the data is outside specific thresholds. The student may have kept some form of notes showing what was tested and the outcome of that testing. Test cases by the student include expected and boundary cases.	The code has been tested for unexpected cases (eg: if an integer is expected, the code has been tested for a decimal <invalid> and string <invalid>). Students may have used 'try/except' code to ensure that their program handles invalid data gracefully. **The student has:** • comprehensively tested and debugged the program. *For example (partial evidence)* The student tests their program in a systematic way to ensure that it works correctly for all logical pathways. Test cases have been well thought out and notes may have been made showing that the code works as expected for all cases. The student has checked all the logical pathways for their program to ensure that it works as expected.

Final grades will be determined on a holistic examination of the evidence provided against the criteria in the achievement standard.

All supporting materials are supplied with this programme and can be found on the TKI website.

ASSESSMENT SCHEDULE: USE BASIC ITERATIVE PROCESSES TO DEVELOP A DIGITAL OUTCOME (91884)

EVIDENCE/JUDGMENTS FOR ACHIEVEMENT/PAETAE	EVIDENCE/JUDGMENTS FOR ACHIEVEMENT WITH MERIT/KAIAKA	EVIDENCE/JUDGMENTS FOR ACHIEVEMENT WITH EXCELLENCE/KAIRANGI
Use basic iterative processes to develop a digital outcome.	Use basic iterative processes to develop an informed digital outcome.	Use basic iterative processes to develop a refined digital outcome.
The student has: • planned a digital outcome to address a problem, need, opportunity or interest. *For example (partial evidence)* The student researches issues relating to quizzes, examines user preferences and plans how they are going to incorporate these through creating sketches and interactive wireframes. The student has used an online tool to plan out the development process. **The student has:** • developed the digital outcome by decomposing it into smaller components. *For example (partial evidence)* The student decomposes their basic computer game into the components that need to be developed and tested such as graphics, functions, user interface, etc.	**The student has:** • used information from testing and trialling to improve the outcome. *For example (partial evidence)* The student provides screen shots with a brief annotation that shows the improvements in the quiz mechanics that were made after making changes. They also provided a short video to demonstrate improved functionality after correcting a bug in the code. **The student has:** • trialled multiple components and/or techniques and selects those which ensure the outcome functions as intended. *For example (partial evidence)* The student trials two different techniques for performing a particular aspect of their quiz and selects the choice that does not cause functionality issues. The student trials two different question generating techniques and chooses the one that is most efficient.	**The student has:** • applied information from planning, testing and trialling of components to develop a high quality outcome. *For example (partial evidence)* • The student has provided evidence that their planning has allowed them to meet project timelines and include all the planned for components and information. Their outcome functions as intended and has no obvious errors in functionality or presentation of the information. Evidence gained from trialling and thorough and organised testing has been integrated into the outcome in an on-going manner to ensure the outcome is of high quality, including aesthetics, functionality and usability.

The student has: • planned and trialled components of the outcome in an iterative manner. *For example (partial evidence)* The student plans and tests the code for the quiz loop. They next plan and test the question and answer part of the quiz. They next plan and test the feedback part of the quiz. The check that quiz is easy to use and understand. Each component is planned and tested in an iterative manner until the final game quiz is produced **The student has:** • tested that the programming outcome functions as intended. *For example (partial evidence)* The student plans testing the functionality of the quiz with various users to ensure the game works **The student has:** • described the relevant implications. *For example (partial evidence)* The student spoke to students from their target audience to determine their quiz preferences. The student recognises that it is unethical to use copyrighted questions. They have recognised that quiz layout will affect the enjoyment of using the quiz. However, the student may not have chosen the best solution to address the considerations or could have more fully addressed these considerations.	**The student has:** • addressed any relevant implications . *For example (partial evidence)* The student addresses that fact that it is unethical to use copyrighted questions by making up their own questions / using creative commons material that has been correctly attributed. They also ensure that any images used in the quiz are copyright free. They have addressed usability and aesthetic considerations through testing their game with a range of end users.	

Final grades will be determined on a holistic examination of the evidence provided against the criteria in the achievement standard.

All supporting materials are supplied with this programme and can be found on the TKI website.