MISSION ENCODEABLE

How to create effective and engaging coding tasks

Festival of Computing June 2026

Anna Wake and Harry Wake (Co-founders)

SUPPORTED BY: Pearson | University of Nottingham | CRAIG N DAVE

Agenda

01 Programming task types

02 Creating engaging scenarios

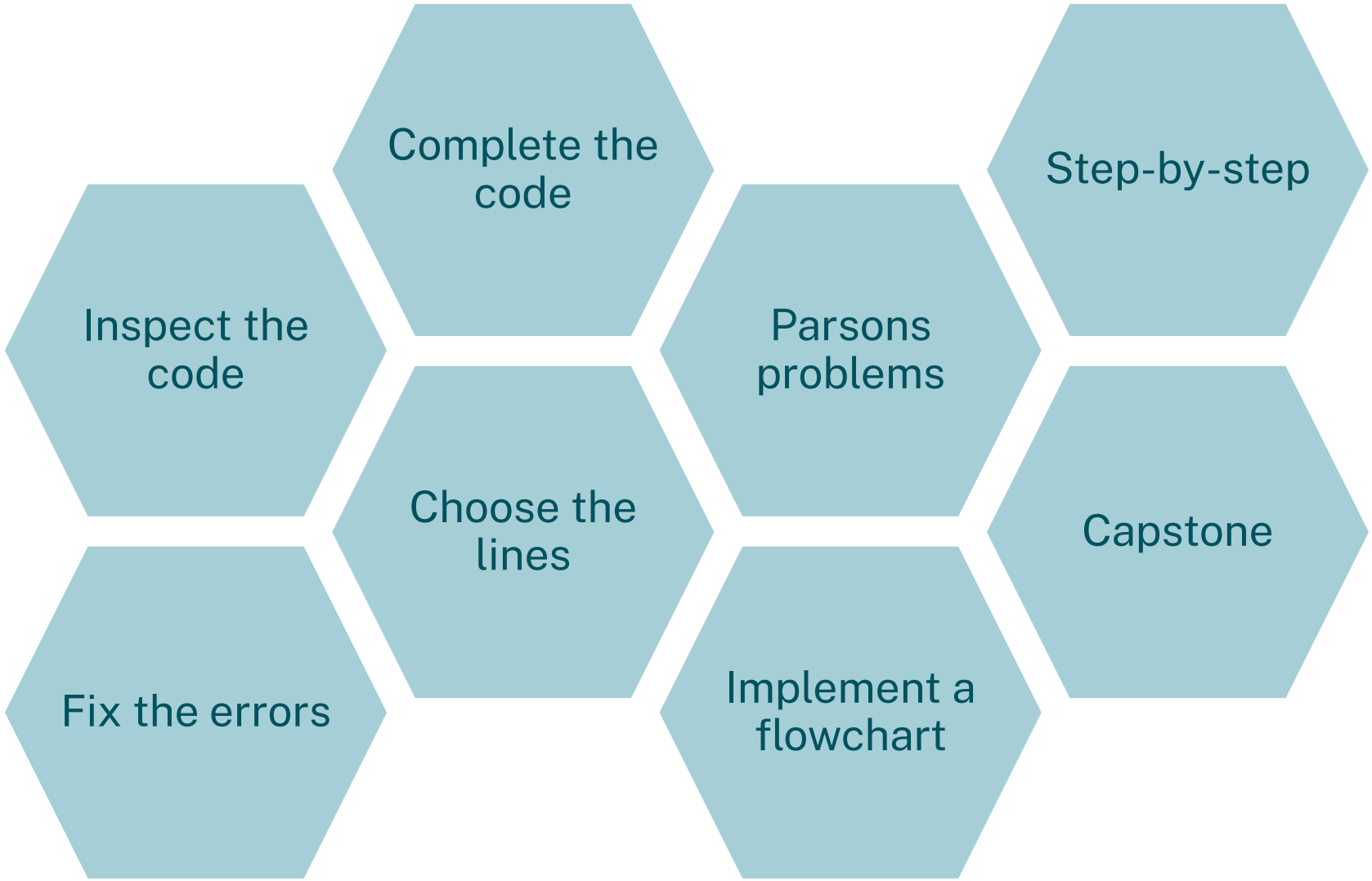
03 Giving feedback on programs

SUPPORTED BY: Pearson | University of Nottingham | CRAIG DAVE

01

Programming task types

SUPPORTED BY: Pearson | University of Nottingham | CRAIG DAVE



Inspect the code

Students must inspect the provided code to find the specified identifier or line number(s).

- Use this as a quick starter task to recap prior learning.
- Helps students build familiarity with technical terms (e.g. selection, count- and condition-controlled iteration), knowledge of which may be required for exams.
- This task type can also be used to create multiple choice questions, enabling quick collection of data.

Take a look at the code below. Give the:

- Line number of a selection construct.
- Identifier of a subprogram.

```
def check_age(age):
  if age >= 18:
    print("You are old enough to vote!")

user_age = int(input("How old are you? "))
check_age(user_age)
```

Mission Encodeable Python Level 2 – Voting Age

Fix the errors

Students must fix syntax, runtime, and logic errors in provided code.

- Use this task type when students have learnt basic syntax and structure.
- Helps students learn to debug code, which is a useful skill for independent projects and more open-ended questions.
- We suggest using common errors in these tasks so that students know to avoid making them.
- Particularly for logic errors, it can be worthwhile to encourage students to work in pairs and explain their reasoning to each other.

We've created a program containing several print statements, which provide some notes about what we've covered so far. Unfortunately, each one of these print statements contains an error.

Debug this code, until you can run it and see no errors.

```
prinnt("Notes from Mission Encodeable")

print(("The print statement is used to output
  strings")

print("Strings are strings of characters"

print(New lines can be created with a \ and
  the letter n)
```

Mission Encodeable Python Level 1 – Note Debugging

Complete the code

Students must complete provided lines and follow instructions to add new lines of code.

- Use this task type to get students writing their own code without needing to start from a completely blank page, which should help to raise their confidence.
- You might want to include “duplicated” lines of code (e.g. in the *Height Chart* program students will write multiple selection blocks), so that students become familiar with the correct syntax.
- You could include comments in the starting code to draw students’ attention to certain lines.

Complete this program by writing an `elif` statement for each 10 cm range between 90 cm and 180 cm.

```
def check_height(height):
    if height < 80:
        print("You're not yet tall enough to
        use this height checker.")
    elif height >= 80 and height < 90:
        print("You are about the same height
        as a snooker table, which is roughly
        80cm tall!")


user_height = int(input("Enter your height to the
  nearest centimetre: "))
check_height(user_height)
```

Mission Encodeable Python Level 2 – Height Chart

Choose the lines

Students must select the correct line of code from four options.

- Use this task type to help students get familiar with syntax and to get them thinking about possible mistakes and the impact they'd have.
- Ideally provide a program with 5-10 different choose the line activities within it so they can find their gaps and consider the overall flow of the program.
- Don't make the answer too obvious - use common mistakes for each of the incorrect options to help students identify and avoid misconceptions.

```
plainText = input ("Enter a message: ")
shift = int (input ("Enter the shift: "))
for letter in plainText:
    # =====> Choose the correct line to check for
  alphabetic letters
    #if (letter.isalnum ()):
    #if (letter.islower ()):
    #if (letter.upper ()):
    #if (letter.isalpha ()):
        value = ord (letter)
        value = value + shift
```

Pearson GCSE CS – Caesar Cipher (June 2024 Q2)

Parsons problems

Students must reorder provided code lines to create a program.

- Use this task type as a starter task or as part of an assessment.
- By removing the need to remember syntax, students can focus on problem solving and sequencing.
- Parsons problems can also be used with flowcharts to help students scaffold their programs.
- You could also consider including “distractor” lines of code that shouldn’t be in the final solution.

Put the code statements into the correct order:

```
state = water_state(temperature)
print("The water is in a", state, "state.")
return "liquid"
else:
return "solid”
def water_state(centigrade):
return "gaseous”
elif centigrade < 100:
temperature = float(input("Enter the
temperature in °C: "))
if centigrade <= 0:
```

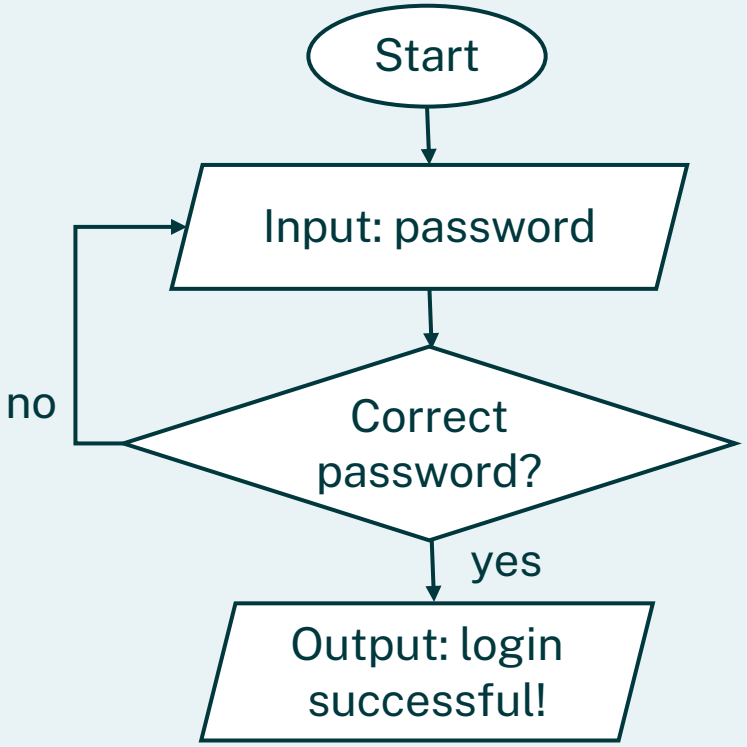
Time 2 Code Python Level 2 – State of water

SUPPORTED BY: Pearson | University of Nottingham | CRAIG N DAVE

Implement a flowchart

Students must convert a given flowchart into a program.

- Use this task type to help students focus on the syntax and increase their fluency with flowcharts.
- This should help students to map from the symbols in a flowchart to the structures in a program.
- You could also try reversing this, where students are given a working program and must draw the corresponding flowchart.



Mission Encodeable Python Level 3 – Password checker

Step-by-step

Students must write lines of code from scratch, using a provided list of instructions.

- Use this task type to bridge the gap to independent programming.
- You should try to structure the instructions so that students understand how they contribute to the overall program, for example, having a section for each subprogram.
- You could include an explanation of more challenging steps, or the syntax of new commands.
- Make the instructions more general as your students become more confident.
- You could write the instructions within the program file as comments.

First of all, print an introduction using a print statement:

```
Welcome to the Mad Libs generator! This project
will generate a silly sentence based on your
responses to the following questions…
```

Then, ask the user for their name, an adjective, a verb, a place, a food, and a vehicle, storing their answer to each question in an appropriately named variable.

Create a variable called sentence. We want the sentence to follow the format:

```
{name} is a very {adjective} person to be around.
They love to {verb} in {place}. Their favourite
food is {food} which they eat in a {vehicle}
```

Finally, print out your new sentence variable and display it to the user.

Mission Encodeable Python Level 1 – Mad Libs

Capstone

Students must design and write a programmed solution to a problem, with little or no scaffolding.

- Use this task type to assess students' independent problem-solving and programming skills.
- Intersperse these regularly throughout the course.
- Make the programs something fun or interesting so that students can enjoy their achievements.
- The aim is for the instructions to be more about what the program should achieve rather than how to build it.

What this project does

An overview of the project, including a sample output, and any other information that might be useful to complete the program.

What you need to do

A breakdown of the steps required to complete the program. This gets progressively less structured through the levels!

Evaluate your program

Specific test cases, or a testing strategy.

Mission Encodeable – Capstone Project structure

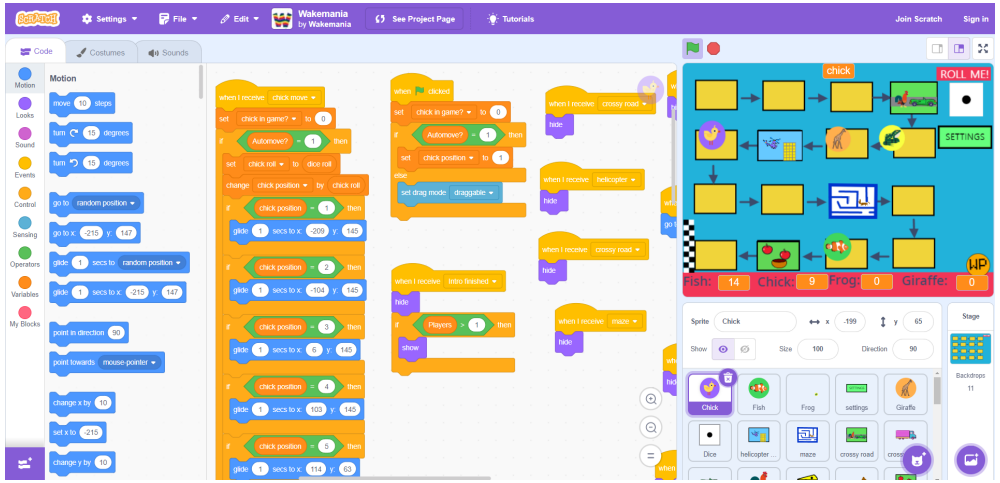
02

Creating engaging scenarios

SUPPORTED BY: Pearson | University of Nottingham | CRAIG DAVE

The progression from Scratch to Python

Students are often taught Scratch through making fun games. This often ends when they learn Python.



A program is used in a shop that sells building materials.

The program reads in data about screws from a file. The data file is provided.

The program counts the number of copper screws.

OCR Security Services is a company that installs intruder alarm systems in commercial buildings.

The systems use a computer that is connected to the door sensors and window sensors.

The following data is stored in the system:

Figure 9 shows an algorithm, represented in pseudo-code, used to display students' test scores. The algorithm does not work as expected and the teacher wants to find the error.

SUPPORTED BY: Pearson | University of Nottingham | CRAIG N DAVE

Inventing engaging scenarios for programs

There are three categories we've identified that can make programs more interesting.

Games

Programs which are competitive, which involve the user making decisions to attempt to reach a goal.

Examples:

- Guess the number
- Hangman
- Four in a row
- Treasure island
- Play your cards right
- BoomBang

Culture

Programs which build students' cultural capital.

Examples:

- Using print statements to display a poem
- Continent bucket list
- Using string formatting to display YouGov survey data / movie ratings

Quirkiness

Programs which feature interesting or amusing scenarios.

Examples:

These sound quite fun:

- Average giraffe height
- Pig Latin

Whereas these sound quite dry:

- Security system simulator
- Curtain price calculator

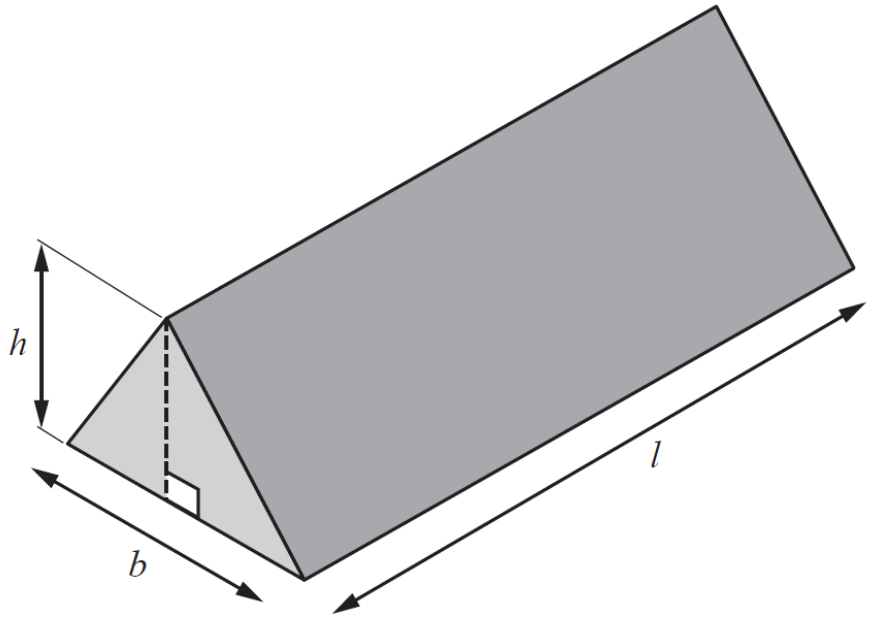
SUPPORTED BY: Pearson | University of Nottingham | CRAIG 'N' DAVE

Transforming a GCSE exam question

Let's consider how to introduce an interesting context to transform this seemingly boring question.

A program is required to calculate the volume of a prism. All dimensions are entered by the user. The dimensions are decimal numbers greater than 0

The volume of this prism is the area of the triangle multiplied by the length of the prism.



Can you think of an alternative scenario for this GCSE exam question to make it more interesting?

SUPPORTED BY: Pearson | University of Nottingham | CRAIG N DAVE

Linking coding exercises to theoretical topics

A great way to consolidate the theory your students are learning whilst also boosting their coding skills.

Algorithms	Data representation	Computer systems	Networks
• Implementation of searching and sorting algorithms, e.g. input an unsorted list of numbers and then display each pass of bubble sort • Output calculator for a Boolean expression with 1-3 inputs	• Binary to denary convertor • Binary shift calculator • Image file size calculator • Audio file size calculator • File size unit prefix convertor • RLE compressor/ decompressor • Bitmap image generator	• Custom PC recommendation tool to compare various hardware components • Disk defragmentation simulator (represent a disk like [['A', 'A', 'B'], ['C', 'C', 'A']] and rearrange into contiguous blocks)	• Network transmission time calculator • HTTP status code lookup • Draw the network topologies using Python Turtle! • Caesar cipher encryption tool • Implementation of Dijkstra's shortest path algorithm

03

Giving feedback on programs

SUPPORTED BY: Pearson | University of Nottingham | CRAIG DAVE

Assessing student work

We recommend both marking the program and assessing how well students understand their code.

Marking a program

- Does the program run?
- Does the program meet all the requirements specified in the question?

Assessing student comprehension

1. **Item:** Can the student identify an example of (iteration/selection etc.) in their program?
2. **Structure:** Can the student explain the syntax of their code? For example, why brackets are needed or why something is indented.
3. **Purpose:** Ask the student to articulate what a few blocks of their code actually achieve.
4. **Reason:** Examine if the program breaks any typical conventions and explore these with the student.
5. **Relation:** Talk through the implications of having structured a program in a particular way and how the lines of code link together. E.g. why a library is included and which line requires it.
6. **Approach:** For more able students and complex problems only, explore their rationale and approach.

SUPPORTED BY: Pearson | University of Nottingham | CRAIG 'n' DAVE

Set actions for improvement

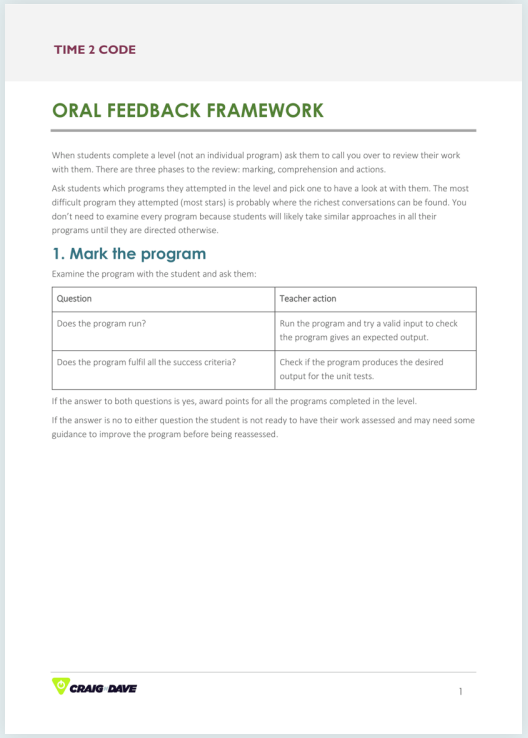
Identify up to two weaknesses in the program and ask students to work on these moving forward.

Select the lowest two in this list, given in order of difficulty, for the student to make the most rapid progress:

1. Include comments to: state what the name of the program is; identify the different sections of a program, e.g. libraries, global variables, constants, subprograms; explain the purpose of selection or iteration statements.
2. Use meaningful identifiers (names of subprograms and variables). E.g. get_input() not inp(), index not i.
3. Use subprograms for different parts of the program. E.g. inputs, the different processes and outputs.
4. Include whitespace between subprograms and longer sections of code to separate them making the program easier to read. E.g. two blank lines between each subprogram.
5. Improve the user experience with more helpful input statements.
6. Use a for loop when the number of iterations is known, not a while loop.
7. Use arrays or lists instead of multiple variables.
8. Prevent the program from crashing when invalid inputs are made.
9. Stop the program ending, offer the user a chance to run the program again or have an infinite loop until the program is closed.
10. Trap exception errors such as file not found.

Oral Feedback Framework from Craig ‘n’ Dave

We highly recommend this four-page reference sheet from our partners at Craig ‘n’ Dave



TIME 2 CODE

ORAL FEEDBACK FRAMEWORK

When students complete a level (not an individual program) ask them to call you over to review their work with them. There are three phases to the review: marking, comprehension and actions.

Ask students which programs they attempted in the level and pick one to have a look at with them. The most difficult program they attempted (most stars) is probably where the richest conversations can be found. You don’t need to examine every program because students will likely take similar approaches in all their programs until they are directed otherwise.

1. Mark the program

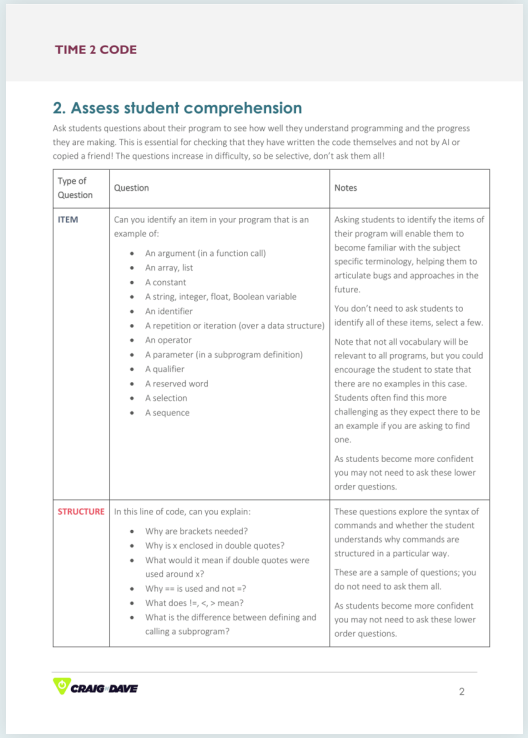
Examine the program with the student and ask them:

Question	Teacher action
Does the program run?	Run the program and try a valid input to check the program gives an expected output.
Does the program fulfil all the success criteria?	Check if the program produces the desired output for the unit tests.

If the answer to both questions is yes, award points for all the programs completed in the level.

If the answer is no to either question the student is not ready to have their work assessed and may need some guidance to improve the program before being reassessed.

CRAIG ‘n’ DAVE 1



TIME 2 CODE

2. Assess student comprehension

Ask students questions about their program to see how well they understand programming and the progress they are making. This is essential for checking that they have written the code themselves and not by AI or copied a friend! The questions increase in difficulty, so be selective, don’t ask them all!

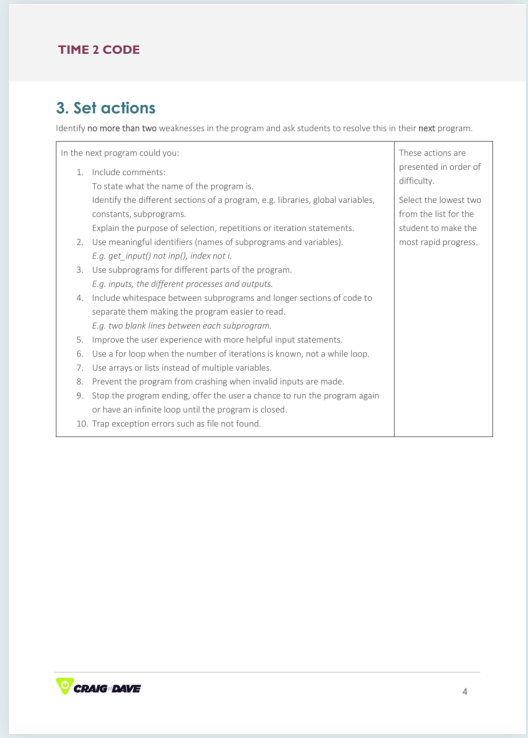
Type of Question	Question	Notes
ITEM	Can you identify an item in your program that is an example of: • An argument (in a function call) • An array, list • A constant • A string, integer, float, Boolean variable • An identifier • A repetition or iteration (over a data structure) • An operator • A parameter (in a subprogram definition) • A qualifier • A reserved word • A selection • A sequence	Asking students to identify the items of their program will enable them to become familiar with the subject specific terminology, helping them to articulate bugs and approaches in the future. You don’t need to ask students to identify all of these items, select a few. Note that not all vocabulary will be relevant to all programs, but you could encourage the student to state that there are no examples in this case. Students often find this more challenging as they expect there to be an example if you are asking to find one. As students become more confident you may not need to ask these lower order questions.
STRUCTURE	In this line of code, can you explain: • Why are brackets needed? • Why is x enclosed in double quotes? • What would it mean if double quotes were used around x? • Why == is used and not =? • What does !=, <, > mean? • What is the difference between defining and calling a subprogram?	These questions explore the syntax of commands and whether the student understands why commands are structured in a particular way. These are a sample of questions; you do not need to ask them all. As students become more confident you may not need to ask these lower order questions.

CRAIG ‘n’ DAVE 2

TIME 2 CODE

PURPOSE	Can you explain what this line of code does? Can you explain what the lines of code between line x and y do?	These questions explore whether a student understands what a block of code achieves. These are particularly relevant for individual subprograms, selection blocks, repetitions and iterations. Often students copy boilerplate code they become familiar with, follow flowcharts or Parson’s problems, but they don’t necessarily know what that code is for.
REASON	Why did you use: • A string instead of an integer, an integer instead of a float? • A while loop instead of a for loop? • Multiple if, elif statements instead of select, match?	Examine if the program breaks any typical conventions and explore these with the student. These questions may not be relevant in all programs but watch for students using one familiar method instead of the most appropriate method, typically with loops and data types of variables.
RELATION	What is the value of the variable x after this line of code executes? What would happen if you moved the line of code from line x to be before/after line y instead? What happens to the value x in the subprogram call when it goes into the subprogram? *I.e. which two variables hold the same value?* Why did you need to include the library x… which line of code requires it?	This is an opportunity to check if the student understands the implications of writing a program in a particular way. Often, they will not understand that a variable needs to be inside or outside of a loop. They also often misunderstand when a flag variable needs to be assigned/reset before entering a condition.
APPROACH	Why did you take this approach to solving the problem?	For more able students and complex problems only, explore their rationale and approach.

CRAIG ‘n’ DAVE 3



TIME 2 CODE

3. Set actions

Identify no more than two weaknesses in the program and ask students to resolve this in their next program.

In the next program could you: 1. Include comments: To state what the name of the program is. Identify the different sections of a program, e.g. libraries, global variables, constants, subprograms. Explain the purpose of selection, repetitions or iteration statements. 2. Use meaningful identifiers (names of subprograms and variables). *E.g. get_input() not inp(), index not i.* 3. Use subprograms for different parts of the program. *E.g. inputs, the different processes and outputs.* 4. Include whitespace between subprograms and longer sections of code to separate them making the program easier to read. *E.g. two blank lines between each subprogram.* 5. Improve the user experience with more helpful input statements. 6. Use a for loop when the number of iterations is known, not a while loop. 7. Use arrays or lists instead of multiple variables. 8. Prevent the program from crashing when invalid inputs are made. 9. Stop the program ending, offer the user a chance to run the program again or have an infinite loop until the program is closed. 10. Trap exception errors such as file not found.	These actions are presented in order of difficulty. Select the lowest two from the list for the student to make the most rapid progress.

CRAIG ‘n’ DAVE 4

https://time2code.today/wp-content/uploads/2023/07/TIME-2-CODE-Oral-feedback-framework.pdf

SUPPORTED BY: Pearson | University of Nottingham | CRAIG ‘n’ DAVE

Thank you.

Anna Wake, Co-founder
anna.wake@missionencodeable.com

Harry Wake, Co-founder
harry.wake@missionencodeable.com

MISSION ENCODEABLE