Introduction

Lesson breakdown:

1. **Voice Recognition. Hello World. Syntax errors.**
2. **Chatbots. Artificial Intelligence. Evaluating expressions**
3. **Comment code. Maths Quiz. 12 x tables quiz.**
4. **Add score. Pseudocode. Responsive feedback**
5. **Making an artificially intelligent Maths game. Assessment**

Teacher Notes

In writing these guidance materials, I am fully aware that highly effective teachers develop their own personal style that they prefer to operate in. When another teacher directs them to follow a list of explicit instructions in a specific sequence, this can constrain the teacher to an unfamiliar style in which they are not accustomed and therefore imposes artificial limits on their effectiveness.
To use these guidance materials to their full potential, the reader is advised to blend, remix, edit and refine these lessons to a style that sits more comfortable. While there is a suggested order to follow, this does not have to be strictly adhered to, and so the parts of lessons have been designed to be modular in approach. When there is value to be gained by deviating from these plans, or spending more time than recommended to enable all learners to progress at an appropriate level of challenge – please do so rather than slavishly follow these plans.

It is also worth stating that purely following these instructions slavishly will not alone teach pupils programming using Python. It is important not to see the activities as tasks that need to be completed, but rather activities to focus learning activities around. An effective, experienced teacher will identify the learning opportunities within each exercise and activity and exploit the potential for learning with these and equally will intervene when pupils are not making sufficient learning progress.
There are 5 lessons worth of content, while some teachers may condense this to 4 lessons others may expand it to fit 8 lessons, all depending on the ability range of the class and the approach adopted by the teacher.

Formatting of text

To make this text easy to follow, I have made use of two some simple styles
Interactive – bold and italic when a specific phrase or technical term should be used
`>>> a = 1` - the three chevrons, precede functions to be typed into the interpreter
`a = 1` – functions with no chevrons are to be typed into the program editor

Why program using Python?

Teachers of computing are incredibly well provided for in that there are currently a fantastic range of resources for teaching programming. Some such as Codecademy facilitate the learning of JavaScript by providing a course of tasks and activities to be followed in a strict sequence, while others like Scratch encourage children to create and share digital stories without needing to worry too much about syntax. There are many who argue in favour of the potential that Greenfoot Java offers and Microsoft Visual Basic equally has many faithful fans. There are many arguments for teaching lower level languages like Java and C#, yet the experiences of many new to these are that while these languages are suited to programming powerful software applications – they are not the most accessible to inexperienced programmers.

The major strengths that set Python apart from these, is that Python is a serious software developer's language but has the distinct advantage in that it is an ideal language in which to learn. Python is particularly suited to handling and manipulating large quantities of data. The fact that Python was chosen for maintaining Google's search quality and YouTube searches demonstrates this programming language's potential to scale to really big projects.
Guido Van Rossum, the pioneer behind Python presented an initiative in 1999 called *Computer Programming for Everybody*, in which he defined his goals for Python:

an easy and intuitive language just as powerful as major competitors

open source, so anyone can contribute to its development

code that is as understandable as plain English

suitability for everyday tasks, allowing for short development times

If you have not read the 'Computer Programming for Everybody' paper, I recommend that you do.

Python is free to download, works on PC, Mac, Linux and Raspberry Pi computers. It seems to be more forgiving regarding syntax errors than many the vast majority of other languages. Programming with Python is a feature of many Computing Science degree courses and was purposely created to facilitate learning to program.

Python is available to download from the Python website http://www.python.org/and documentation is available here including tutorials http://docs.python.org/py3k/
There is a huge range of materials available online to learn programming with Python on this page here http://wiki.python.org/moin/BeginnersGuide/NonProgrammers It is recommended that you use some of these to supplement these lessons in class and for homework.

Teachers Beware: Common Errors
From my experiences of teaching programming with Python from Year 7 to Year 11, I have encountered some common errors that children experience. Hopefully, if I describe the most common ones here – it may save you some sweat and tears!
Syntax Errors– These are not unique to Python, but less frequent an occurrence. This can include capitalising variables in one instance, and not in another – Python will treat 'Lives' as being a different entity from 'lives'. Wrongly using capital letters for functions; it is the norm in Python for functions to be written in all lower case, e.g. print, input. Pupils may omit colons, and brackets when they are required. Some pupils have tried using two single quotes `''` instead of a double `"`.
Indents – Many other languages use curly brackets {} to denote blocks of code. However, Python uses indents to separate blocks of code.
Saving .py extension – It is a minor frustration that early versions of IDLE (the development environment) do not automatically append **.py** to the end of the filename. In the latest version this has been fixed so that it is not necessary each time when saving a program to add .py at the end of the filename. When pupils have not done this, one obvious clue is that their code does not feature syntax colour coding.

Version of Python

From time to time, a new version of the Python language is introduced. There were some fairly significant changes between Python 2.0 and Python 3.0. At the time of writing, the current version is 3.3.0 You should download the latest version to ensure compatibility with these materials.

Lesson 1

Learning Outcomes - By the end of the lesson, pupils should be able to:
Evaluate the effectiveness of Siri on the iPhone and describe
Create a "Hello World" program in the Python
Identify and correct common errors in Python programs
Create, save and test a 'Hello World' program

Part 1
Evaluating the effectiveness of speech recognition technology
Please explain to the class that over the next couple of lessons, they are going to attempt to create a computer program that can think and reply to questions much like 'Siri'. First, however – it is important to have a shared understanding of products like Siri that make use of speech recognition technology and their limitations. At this point, either arrange for a demonstration of Siri or equivalent, or select the most appropriate from the suggested YouTube clips.

Leading the class through a ***think, pair, share*** activity. You might start by asking pupils to share their current impressions of Siri technology and how much this adds to the desirability of devices like iPhones. Alternatively, wait until watching one or more of these short video clips, or other similar material. Reserve showing 'Eleventh Floor' until afterwards for some light-hearted rumour. Advice - please preview first to judge appropriateness of language used.

Following the video clip(s) ask each pupil to first think of (or list) the advantages and disadvantages of technology like Siri and how it may advance 10 years from now. After some short thinking time, ask them to then share & compare their list with their partner, noting differences on their response sheet. Then forming groups of 4 or more they should collate their collective responses ready to share these with the rest of the class. To keep to time constraints, each group could report one feature back to the whole class. There is potential here for a homework related activity in which pupils write their own summary of the whole class's thoughts about Siri. This might be an ideal time to play the 'Eleventh Floor' clip which highlights one major criticism of speech recognition technology.

Resources

Siri iPhone 4s	http://youtu.be/5mNcnj2l6RE
Siri Demonstration	http://youtu.be/MpjpVAB06O4
Siri talks to Siri	http://youtu.be/XBRXA8zmJr8

Search for the "Eleventh Floor" video on YouTube showing two Scots in a lift. Due to copyright, it may be moved or taken down. It is worth seeking out if you can find it. It's an excellent, humourous demonstration of 'syntax errors'
Caution: There is some language which may not be suitable for all ages; I recommend you preview the video first.
Part 2
Using the interactive mode with Python
Before creating an artificial intelligence simulation, some Python fundamentals. For the next activity, pupils will need to locate and open the Python IDE called 'IDLE' (trivia: named IDLE after Eric Idle). It might be appropriate to show the class what it looks like.

Explain that there is a tradition when learning a new programming language to first create a Hello World program. Point out the location of Python's ***command prompt***, (the triple chevron >>>) and identify it as such. Ask pupils to type in the following, exactly as it shown here and then press the return key. Note that functions in Python are always in lower case.

```
>>> print("Hello World")
```

The phrase *Hello World* should appear immediately below the print as shown below

```
>>> print("Hello World")
Hello World
>>>
```

Part 3
Understanding what Syntax Errors are and how to avoid them

Now ask pupils to type the following in, exactly as it shown here and then to press the return key.

```
>>> print(Hello World)
```

The following error message will appear.

```
SyntaxError: invalid syntax
```

Computers are machines that are very literal when it comes to following instructions; humans are rarely so literal. Computers are not as good dealing with *nearly, almost*, and *not quite* in the way that we humans are. ***Syntax*** is used to describe the rules that determine the way that instructions and commands must be written. Python is reputed to be more forgiving of syntax errors than many other programming languages, which can make it easier to learn, meaning more time can be spent creating than ***debugging***. IDLE, the Python IDE has ***syntax highlighting*** which automatically assigns colours to different elements, e.g. the "Hello World" phrase should be green. This should help spot some simple errors when typing in commands.

To gain a better grasp of syntax errors, ask pupils to first predict what will happen with some of the following and then try them to see what if the response they get matches up with what they predicted.

```
>>> print"Hello World"
>>> print("Hello World");
>>> Print("Hello World")
>>> print("Hel World")
>>> prin(Hello World)
```

Some will work and others won't, ask pupils to also find out and try their own variations. Ask pupils if they can identify a pattern to predict what variations work and which ones do not. Initially, this might seem a fool's errand – but some groundwork here will reduce debugging problems later on if pupils do not have an understanding of the concept of syntax errors.

Guidance - One important point to highlight is that in the ***interactive mode***, instructions are executed straight away. It is perhaps worth pointing out as well that 'print' in programming usually means on screen, not on paper.

Part 4
Create, save and test a 'Hello World' program
During this final part of the lesson, pupils will learn how to create, save and execute a "Hello World" program using Python.

Demonstrate opening a new window, by choosing the File menu in IDLE and selecting New Window (Ctrl + N). In this ***file editor*** mode, it is possible to create programs that the interpreter will not execute straight away, but only when called or executed. One of the obvious differences between the ineractive mode and the editor is the lack of a command prompt.

Guidance - It is possible that pupils may confuse the two different windows, so an explanation would be advised. The analogy of a shopping list may help explain, in the interactibve mode if you said "Buy milk" it would do it straight away, then "Buy eggs" etc, however with the editor it is more like making a list of items to buy which only work when you say "run shopping list". The advantage of a list is that you can test it, make changes to it, save it for use again.

In the Python programming editor, ask pupils to type the following in, exactly as shown. If they are not familiar with the ***underscore key*** or the ***double quotation mark*** they may need guidance. Watch out for pupils who try to use two single quotation marks instead of the double.

```
print("Please type your name in")
my_name = input ()
print("Nice to meet you " + my_name)
```

Then pupils will need to save their program using the filename **my_name.py** in their My Documents or other appropriate location. (If not using Python 3.3.0, please make it explicitly clear that they must always add the .py extension on the end of their filename when saving first time)

Guidance – Some potential problems at this stage are that Python when saving a program will often default to the Python directory as a location for saving files, so pupils will need to check carefully rather than simply clicking 'Save'. Also, in earlier versions, the Python IDLE does not automatically add the filename extension **.py** on the end, it is necessary to append this onto the file name. You will find that many pupils will forget to do this with the following results – their syntax highlighting in colours will return to all black and to add to their woes, when they try to open a file it will seem to have disappeared. If this happens they will need to locate the file using Windows Explorer or looking in their My Documents folder and then renaming it with .py on the end.

Once their program module has been saved, they can run it by either pressing the F5 key or going to the Run menu and selecting Run Module. If time allows, ask pupils to add more questions into their my_name program by copying the first lines of code, pasting them and changing the question text.

Lesson 2

Learning Outcomes - By the end of the lesson, pupils should be able to
Describe an experience of Artificial Intelligence from using chatbots
Build the first part of an artificial intelligence program using Python
Use the Python Interpreter as a calculator and explain what an integer is

Part 1
Describe an experience of Artificial Intelligence from using chatbots
To develop a better understanding of some current artificial intelligence technologies like Siri, allow pupils 5 – 10 minutes to allow them opportunities to try some of these themselves. Warn pupils that some of these chatbots listed in the resources are experimental models. Provide a list of prompts for pupils to consider. Then invite pupils to share their answers among themselves or with the rest of the class.

1. What sorts of questions are the chatbots very good at?
2. What questions do chatbots not answer very well?
3. What process/stages are taking place after the user types in a comment?
4. What tricks does the chatbot use to make it seem real, e.g. artificially intelligent?

Resources
A chatbot that talks with sound http://chaturing.com/artwork/ select the 'chatbot' menu
A simple text-only chatbot http://chaturing.com/artwork/chatbot/
iGod http://www.titane.ca/concordia/dfar251/igod/main.html
Rosette http://labs.telltalegames.com/rosette/
A directory of chatbots http://www.chatbots.org/language/english/
Another directory of chatbots http://www.pandorabots.com

Part 2
Building the first part of an artificial intelligence program using Python.

In the last lesson, pupils should have created the program module below and saved it as my_name.py. This three line phrase can be developed into a very simple Artificial Intelligence program.

```
print("Please type your name in")
my_name = input ()
print("Nice to meet you " + my_name)
```

Considering the kinds of questions and information that people ask about each other when they first meet, ask pupils to **think, pair and share** the topics they considered. Some of these may include where they live, if they have brothers and sisters, what age they are, favourite food etc. At the end of the activity, each pupil should have a list of question topics.

A short demonstration may be needed of copying the first 3 lines of script, copying and pasting it and editing the content to match a new question as in this example. Suggest to pupils that they save this with the name questions.py – remembering to add the .py filename extension.

```
print("Please type your name in")
my_name = input ()
print("Nice to meet you " + my_name)
print("So, " + my_name + ", what is your favourite food?")
favourite_food = input ()
print("Ah, your favourite food is " + favourite_food)
```

Next pupils should consider adding more complex questions into their script. Notice in the 4th line above it is necessary to have two + signs, one either side of the variable my_name. Common errors to watch for are pupils not having a complete pair of double quotation marks or adding + signs between ***variables*** and ***strings***.

A ***variable*** is a stored value, a ***string*** (or text string) is a sequence of characters which may be words. This would be an ideal opportunity to identify to pupils what a string is and what a variable is – perhaps ask them to search for definitions and develop their own explanation, or identify them in their own program module. A string can be stored in a variable.

A suitable challenge to test understanding of this activity would be to create some script at the end of the program module that summarises e.g. *"So Alan, it was lovely to meet you. I now know that you live in Preston and that your favourite food is Pizza"*.

Collaborative Learners - There will be a lot of value in asking pupils to swap places and evaluate each other's programs. You could agree awarding of points for levels of challenge, e.g. program that works =1 point, working program with 3 or more questions =2 points, working program with 3 or more questions and summary at end =3 points. If you set another related challenge – this would allow you time to reward those with 3 points for their achievement, identify who is struggling 1 point or less, or ask those with 2 points or more to support pupils with 1 point or less.

Part 3
Using the Python Interpreter as a calculator
Going back now to the imperative mode of the interpreter, we see that Python can function as a calculator. Ask pupils to solve a series of maths problems using the interpreter, but do not reveal how to do it straight away.

Activity – Ask pupils to find out how to use Python to work out the answers to these maths problems. 156 add 567, 132 subtract 46, 256 divided by 8, 389 multiplied by 13. At this early stage, avoid explaining the keyboard symbols for ***mathematical operators*** to see if the class can work them out for themselves.

The method is shown below, but do not reveal it yet, instead ask the class if they can work it out.

```
>>> print(2 + 2)
```

The answers are:

```
>>> print(156 + 567)
723
>>> print(132 - 46)
86
>>> print(256 / 8)
32.0
>>> print(389 * 13)
5057
```

Please explain to the class that the ***mathematical operators*** in computing are add +, subtract – as they would expect, but multiply is * and divide is / . This activity goes a long way to explain the strengths & weaknesses of computers. An effective way to check understanding would be to ask the class to explain why the following do not work. You might ask them to share this with a partner or larger group before sharing with the whole class.

```
>>> what is 2 add 2?
>>> 3 times 6
>>> 24 subtract 3
>>> how many times will 4 fit into 12?
```

In computing, whole numbers (without decimals) are referred to as ***integers***, this means that while 4.0 is not considered an integer, 4 is. It is possible to store integers into ***variables***.

A short activity to end the lesson could be to assign numbers to variables as in this example below.

```
>>> pizza = 250
>>> coke = 100
>>> chips = 150
```

Some interesting expressions can then be evaluated such as:

```
>>> pizza + chips
400
>>> 2 * pizza
500
```

The variables could be pupils' names and ages. Learning this will form the foundation for the next lesson.

Lesson 3

Learning Outcomes - By the end of the lesson, pupils should be able to
Demonstrate and explain the practice of commenting their code
Create a simple maths test, explain the need to convert a string to an integer
Make use of 'if' statements to create a 12 times table test

Recap
In our last lesson pupils used the script below to create the foundations of an artificial intelligence program. They should have managed to add some questions of their own.

```
print("Please type your name in")
my_name = input ()
print("Nice to meet you " + my_name)
```

A useful way to go back and check pupils' understanding of all this would be (with the minimum of help) to ask all the class to locate and open their questions.py program and add another question, this time asking the user to name a place they have been on their holidays and then respond with an answer using the name of the holiday location, e.g. *"Hmmm..." + holiday_location + " sounds a like a nice place to go"*

Comment Your Code
This is now a good time to introduce the practice of ***commenting code*** to the class. Comments are often added to computer programs to allow people to understand the intentions of the person who created the code. The Python interpreter ignores the comments completely, so syntax is not a problem. In practise, it is not necessary to comment every line/section – only where it is not obvious what is going on. However, as the class have probably not experienced commenting before, we will apply some comments to the simple code we have written so far using the hash key (#). You will notice that when you use the hash key in the program editor, the text changes to red to indicate the use of commenting. Comments can be used for sections, or in-line as shown below.

```
# This program finds out the user's name
print("Please type your name in") #prints message to user
my_name = input () #stores user's name in a variable
print("Nice to meet you " + my_name) #displays message
```

Activity – Ask pupils to add comments to all their lines of code. Some questions for pupils to consider. Try using the think – pair – share approach with these questions.

1. **How could you add some information about the program, creator, date etc.?**
   Answer- add a top line comment with name, date etc.
2. **Is it necessary to comment on every single line?**
   Answer- Only comment when it is not clear to another person
3. **If the interpreter ignores everything after the #, how else could this be useful?**

Answer- You can use # to 'comment out' sections of code that are not working to help with debugging.

Creating a maths quiz

There is a lot of value in creating games when learning how to program. This part of the lesson starts with a simple question script that when understood can be used to build a more sophisticated game. Start by asking pupils to open a new program editor window, we do this in the interpreter by selecting File – New Window. Then ask them to create the script below, exactly as it appears.

Please pay particular attention to the use of spaces, indents and colons. Some programming languages, Java for example make extensive use of { } brackets to mark blocks or phrases of code. Python uses indents instead to mark out separate blocks. So, it is crucial that the indents are used appropriately.

Ask pupils to save this as **maths_question.py** – if they all use the same name, it will make it easier in future lessons to retrieve their work.

```
print("What is 2 + 2?")
answer = input ()
answer = int(answer)

if answer == 4:
    print("Well done")

else:
    print("Sorry the answer was 4")
```

Some will complete this more quickly than others, so ask them to add more questions to their game while waiting. Once the class have all tested their games, debugged them and run them successfully, we need to see if the pupils can explain some of the new things in this program.

1. **answer = int(answer) - thinking back to use of integers last lesson, what does this do and what does 'int' mean?** Answer- this converts a text string into a number or integer. If we did not convert it to an integer, we could not compare it to another integer, the answer 4.
2. **If answer = = 4: - What does the '= =' translate to in plain English** Answer- this means 'equal' to, as "in is it equal to?"
3. **What does the 'else' translate to in plain English?** Answer-this translates as 'or else'
4. **Why are the indents necessary after the if and else statements?** Answer- this means follow these instructions if the statement above is true
5. **What happens if the colons are not there after the 4 or else?** You get a Syntax Error

Part 3

To check pupils' understanding of everything learnt so far, ask pupils to create a test that will check the user's knowledge of the 12 times multiplication table, with between 4 and 12 questions. Explain that they must save it with the name **12_times_table.py** and it must include a header comment, with description, name of creator and date. The program must feature some other use of comments and must work successfully.

Although there is an assessment piece being set in the 5th lesson, you might decide that this task be used to provide the teacher with some evidence for assessment of the last three lessons. If this is likely to be used as a formal assessment piece I suggest you share some assessment criteria with pupils before they start.

Lesson 4

Learning Outcomes - By the end of the lesson, pupils should be able to
Demonstrate use of variables to add score to their game
Designing a coding solution using Pseudocode
Add responsive score feedback to their game

Part1
Demonstrating use of variables to add score to their game
In the last lesson, pupils used the following script to create the foundation of a maths quiz and then developed it into a 12 times table test.

```
print("What is 2 + 2?")
answer = input ()
answer = int(answer)

if answer == 4:
    print("Well done")

else:
    print("Sorry the answer was 4")
```

With just a few lines of extra code, it should be possible to maintain a score throughout a game and then give the user some feedback at the end. There may be some value in asking pupils to type the code in again from scratch, getting them used to practice of typing in code. However, if pushed for time – you could just ask them to open their **maths_question.py**

Then**,** ask the class to modify their script thus, first line and last line. Check that pupils understand the concept of *score now equals itself plus one*.

```
score = 0 #this defines variable score, and sets it as zero
print("What is 2 + 2?")
answer = input ()
answer = int(answer)

if answer == 4:
    print("Well done")
    score = score + 1 #this increases score by one

else:
    print("Sorry the answer was 4")
```

In addition, pupils will need to add a facility to display the score at the end of the game (print score) allow them to decide how to do this, or identify it is missing. A simple solution to print the score at the end of the game would be to add this line to the bottom of their program, with no indent;

```
print(score)
```

However, a more sophisticated way to do this would be to add the following.

```
print("Your score was " + str(score))
```

A worthwhile exercise would be to ask pupils why it is necessary to write str(score) or ask them to try the following

```
print("Your score was " + score)
```

They should find that, because the score variable is an integer, Python cannot simply add it on to the end of a string. So the str( ) function temporarily converts to a string for the purposes of this line. Python cannot add strings and integers in their raw form together when printing.

As an extension to this, ask pupils to locate and open their **12_times_table.py** program from last lesson and add a scoring feature to each question. Finally ask them to add a line that prints the score at the end. You could suggest that some 'mean people' might also deduct a point for every wrong answer and add this to their script.

Part2
Planning a coding solution using Pseudocode
Rather than simply report the score that the user has achieved, it would seem more impressive to give an appropriate response based on how high or low the user score was. Before we look at how to code this solution, now would be a good time to understand what Pseudocode is.

To introduce the concept of ***Pseudocode*** to the class, you might ask them to **think, pair and share** their own explanations first of all to help you establish what they think Pseudocode might be. Alternatively, you might simply ask them to find out using an internet search.

Definition: Pseudocode is a name given to a technique for designing and planning a coding solution that does not involve using the actual code or programming language. Indeed, while aspects of it may look like a programming language to a non-programmer – there may be too many syntax errors for it to actually work. The aim is to enable someone to code a solution without being distracted by the actual syntax and therefore think a little more creatively. To help pupils gain a better understanding of the principle of Pseudocode, describe how one of the programs from Lesson 2 functioned, and without showing them - ask them to write on paper (with a partner) the Pseudocode for:

```
print("Please type your name in")
my_name = input ()
print("Nice to meet you " + my_name)
```

The pupils' Pseudocode solutions could look something like this:
print "type your name in"
input user name
print "nice to meet you" + user name

If they have not quite grasped the concept of pseudocoded solutions yet, try challenging them with some of the other blocks of code they have created, e.g. the block of code that asks the question

"What is 2 + 2?" and then checks for the correct the answer. It may be necessary to work through a solution with them until it makes sense.

Part 3

Add responsive score feedback to their game, i.e. Score greater than, less than

Rather than simply report the score that the user has achieved, it would seem more impressive to give an appropriate feedback response based on how high or low the user score is at the end of the game.

Ask pupils working in pairs to Pseudocode a solution to this. It could work with two simple outcomes. If it is possible to score a maximum of 3, you could have a specific response for a score of 2 or more out of 3 and another response for 1 or 0.

Once pupils have written their Pseudocode solutions, ask them to try writing them in Python. It is inevitable that some pupils' solutions will fail, this is a good outcome – since there is so much more potential to be gained from fixing a failed solution than a solution that works first time.

This solution below, sets the score to a fixed value for the purposes of testing. If pupils are really struggling to make their solution work, you may choose to share part or all of this solution with them to help.

```
score = 0
if score > 2:
    print("Well done!")
else:
    print("Oh dear")
```

Once the actual mechanism of the solution is working, encourage your pupils to modify the feedback messages to a more human response, e.g. *"Well done, you scored 3 out 3 – very clever", "Oh dear, you only scored 1 out of 3 – why not try again to improve"*

Lesson 5 - Assessment

Learning Outcomes - By the end of the lesson, pupils should be able to
Design and code their own maths game based on the learning from the previous 4 lessons

The task
Starting with a blank file, you need to create your own number game using Python. You should add an appropriate amount of comments to your game including a header with your name, the title and a brief description of how to play the game. You should also add in line comments to explain particular parts of the program code. Your game should include at least 5 questions (change as appropriate) and you need to decide what the actual questions are about and the level of difficulty.

Examples
The teacher may give examples of the questions, apart from some obvious maths questions – other examples may include:
How many seconds are there in 1 hour? How many days are there in a leap year?
If a triangle has two angles at 45 degrees, what will the remaining angle be?
If a = 2b -5 and b is 6, what is a? What is the minimum legal age to vote in the UK?
What is 10010001 converted to denary?

Success Criteria:

All of the class must
Create a game that works and save it.
Use enough comments to explain how the game works
Add a scoring feature to the game.

Some of the class may
Ask the user some questions at the beginning of the game, e.g. user's name and make use of it throughout the game.
Add a function to report the score after each question.
Include some responsive feedback at the end of the game specific to the score
In a comment describe a feature to be included in a future version of the game

A few might
Include a larger set of questions that increase with difficulty as the game progresses
Allow the user to try again on a wrong answer, but award a reduced score for a second try
Give the user some feedback mid way through the game as encouragement.
Add some comments explaining how they could add extra features to the game

Teachers are encouraged to modify the success criteria above to suit the ability of their class.

I would also suggest that you use some peer marking initially after completion of this assignment to give the class some prompt feedback. You might do this by working through one example on a projector/display and explain how marks/grades are awarded. Then ask pupils to swap seating and grade each others' games. It would be a good idea to let pupils improve their games once they have received some feedback from their peers.