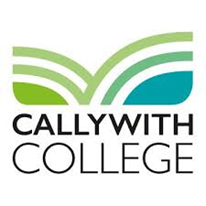
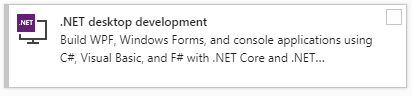
** A Level Computer Science**

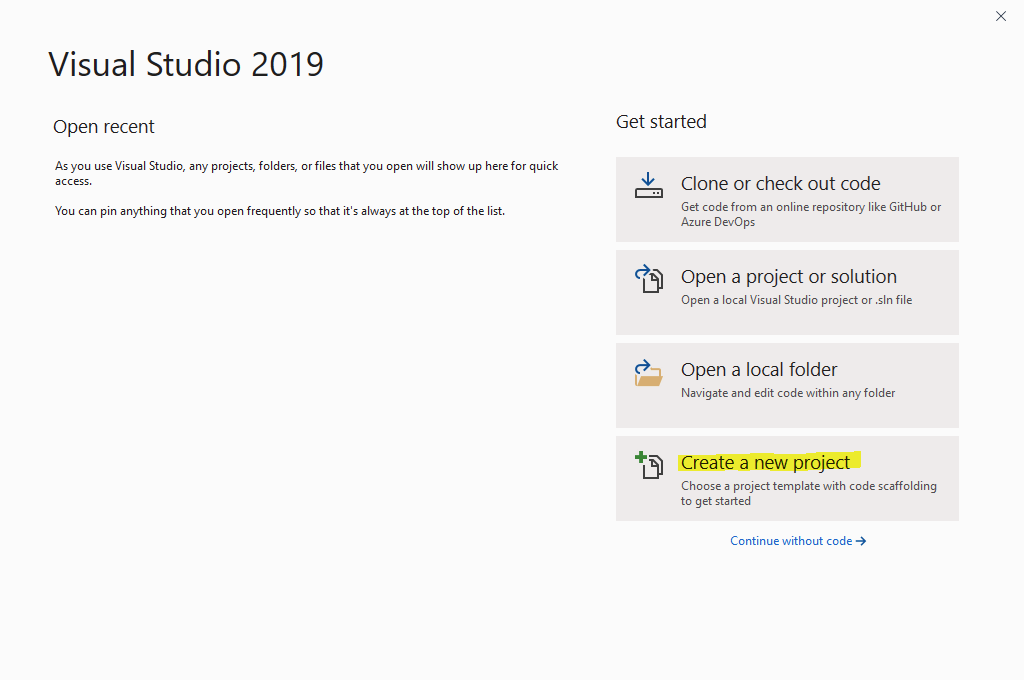
On the following pages you’ll find programming theory and several challenges. You’ll need to read through the theory, try out the sample code provided, and then have a go at the challenges. Do feel free to do extra research when working through the challenges, such as through Google and Stack Overflow. The primary language for A level is Visual Basic and I would recommend you use this. If you are very confident with programming and would prefer to use C#, then you could use this instead.

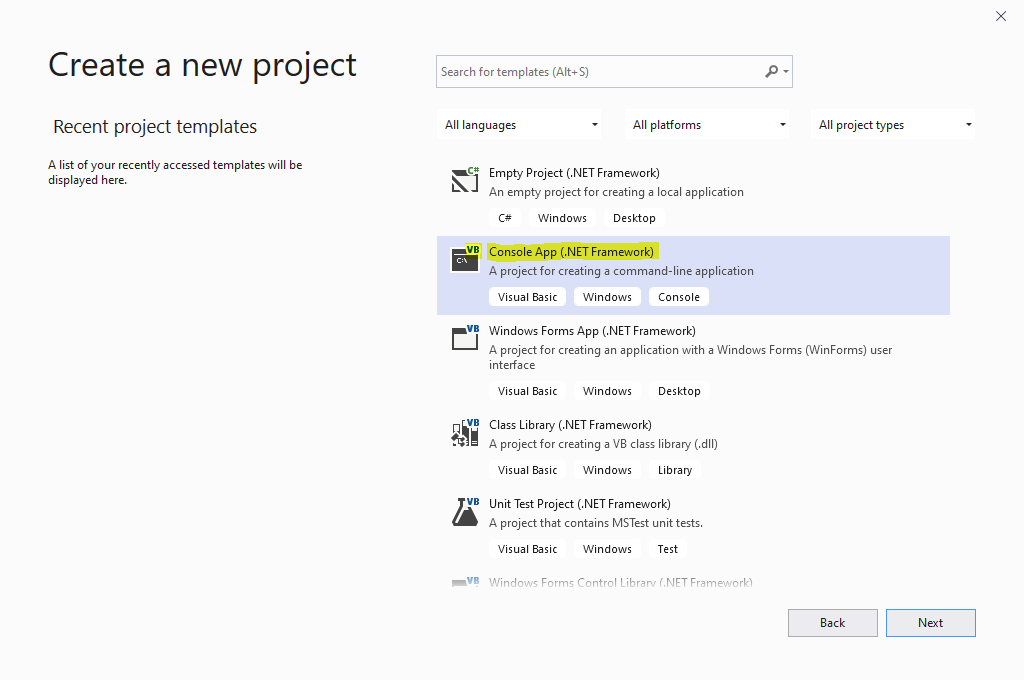
Before you get started you will need to install "Visual Studio Community Edition" which you can download for free here: <https://visualstudio.microsoft.com/downloads/>

When installing Visual Studio, select this option:



To create a new project:





**Now it’s time for some theory….**

# Getting started in console Visual Basic

1. Open Visual Basic Express or Visual Studio Express. This guide is written for versions 2008-2015.
2. Choose File… New Project…
3. Choose Visual Basic… Console Application…
4. Make sure you name your program appropriately, e.g. “Objective 1 Task 1” and choose an appropriate place to save the project. Note, Visual Basic programs consist of many individual files grouped into a solution and therefore you should enable ‘Create directory for solution’ (default).

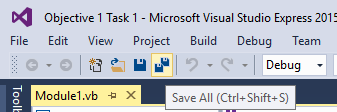
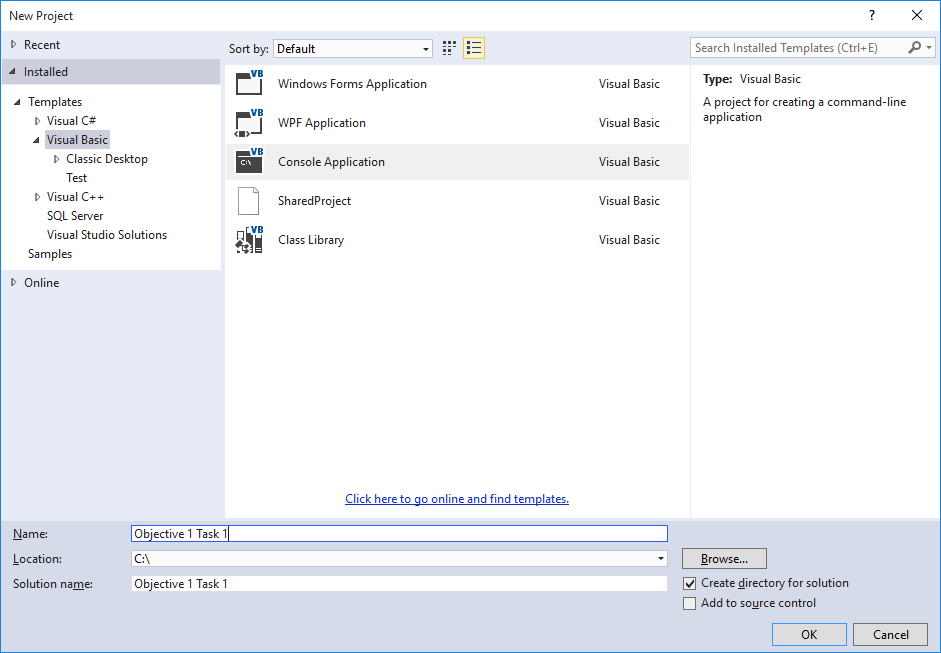
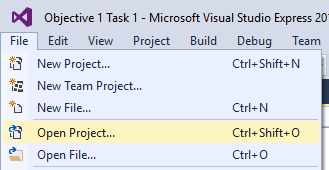


Figure 1: screen shot from Visual Studio Express 2015

1. To run your program, press F5.
2. When you save your work, make sure you choose Save All
3. When you want to open your work again, choose Open Project…, **not** Open File. Find the .sln file.



1. When working on challenges, remember to put Console.Readline() at the end of your program, otherwise the code will run, and end before you can see the output!

# Objective 1: Understand how to output text strings

In this objective you learn how to output text to the screen.

## Tasks

1. Try entering the following commands and see what happens:

Console.WriteLine("Hello World")

Console.ReadLine()

1. Try entering the text without the speech marks and see what happens:

Console.WriteLine(Hello World)

Console.ReadLine()

Note the difference. One works and one doesn’t! If you want to output text to the screen it must be included in speech marks. Text is called a string in programming and must be enclosed in double quotes.

1. Try entering the following commands and see what happens:

Console.WriteLine("Hello World" & " this is my first program.")

Console.ReadLine()

Note how an ampersand symbol (&) joins strings together. This is called concatenation.

1. Try this sequence of instructions:

'Start of message

Console.WriteLine("Hello World" & " this is my first program.")

'Blank line

Console.WriteLine()

'End of message

Console.WriteLine("I am learning to code...")

Console.WriteLine("...and it is fun")

'Wait for user to exit

Console.ReadLine()

Note the apostrophe symbol enables you to insert a comment in the code. The computer ignores the comments, but they help to introduce the program and allow you to leave notes in the code to make it easier to understand later.

1. Change the program so it outputs this instead:  
     
   Computers only do exactly as they are told…  
   …so you need the code to be correct!

If you make any mistake with the commands, it won’t work

# Objective 1: Key learning points Understand how to output text strings

* Text is known as a **string** in programming and must be enclosed in double quotes.
* Strings can be joined together using an ampersand symbol. This is known as **concatenation**.
* When a program is run it is called **executing** the program.
* A set of instructions that execute one after another is known as a **sequence**.
* Comments can be inserted into the code using an apostrophe symbol. These are helpful to make notes within the program so that the code can be understood again at a later date or by another programmer.

# Objective 1: Key words

### Console.Writeline()

Example code: Console.Writeline(x)

Purpose: to output x to the screen followed by a carriage return.

### Console.Write()

Example code: Console.Write(x)

Purpose: to output x to the screen without a carriage return.

# Objective 2: Understand how to input strings and numbers into variables

In this objective you learn how to get input from the keyboard to use in your program.

## Tasks

1. Try entering the following commands and see what happens:

'Declare variables

Dim name\_entered As String

'Inputting strings

Console.WriteLine("Hello")

Console.WriteLine("What is your name?")

name\_entered = Console.ReadLine

Console.WriteLine("Thank you " & name\_entered)

'Wait to close the program

Console.ReadLine()

1. Try entering the following commands and see what happens:

Dim year As Integer

Console.WriteLine("What year is it please? ")

year = Console.ReadLine

Console.WriteLine("Ah, it is " & year & " thank you.")

Console.ReadLine()

1. Change the program so it asks you for your name and your age, outputting for example:  
     
   Thank you Dave. You have registered an age of 15.

# Objective 2: Key learning points How to input strings and numbers into variables

* Data is input by a user into a **variable**.
* Memory is reserved for variables using the Dim command. This is known as **declaring** the variable.
* Variables have a data type: string, integer or decimal as examples, indicating how much memory they will use and the type of data they will store.
* Variables must be declared before they can be used.

# Objective 2: Key words

### Dim

Example code: Dim x As y

Purpose: to declare a variable to be used later in the program.

x is the name of the variable. y is the data type for the variable. y can be string (text), integer (whole number), decimal (number with decimal places) or date.

### Console.Readline

Example code: x = Console.Readline

Purpose: to store text input at the keyboard into a variable, x which can be used later in the program without inputting again.

# Objective 3: Understand string manipulation functions

In this objective you learn how to extract and change data that has been input.

## Tasks

1. Try entering the following commands and see what happens:

'UCase changes a string to upper case

Dim forename, forename\_uppercase As String

Console.Write("Enter your surname: ")

forename = Console.ReadLine()

forename\_uppercase = UCase(forename)

Console.WriteLine("Your name in capital letters is: " & 🡪 forename\_uppercase)

Console.ReadLine()

1. Change the program to ask for an email address, outputting the string in lowercase using LCase() instead of UCase.
2. Try entering the following commands and see what happens:

'Len returns the number of characters in a string

Dim forename, length\_name As String

Console.Write("Enter your surname: ")

surname = Console.ReadLine()

length\_name = Len(surname)

Console.WriteLine("There are " & length\_name & " letters in 🡪 your name.")

Console.ReadLine()

1. Try entering the following commands and see what happens:

'Left returns a number of characters to the left of a string

Dim sentence, characters As String

sentence = "I saw a wolf in the forest. A lonely wolf."

characters = Left(sentence, 5)

Console.WriteLine(characters)

Console.ReadLine()

1. Change the program to output the last 12 characters in the sentence using Right() instead of Left().
2. Try entering the following commands and see what happens:

'Mid returns a number of characters in the middle of a string

Dim sentence, characters As String

sentence = "I saw a wolf in the forest. A lonely wolf."

characters = Mid(sentence, 21, 6)

Console.WriteLine(characters)

Console.ReadLine()

1. Try entering the following commands and see what happens:

'Instr returns the location of one string inside another

Dim sentence, word As String

Dim position As Integer

sentence = "I saw a wolf in the forest. A lonely wolf."

Console.WriteLine(sentence)

Console.Write("Enter the word to find: ")

word = Console.ReadLine

position = InStr(sentence, word)

Console.WriteLine("The word " & word & " is at character " & 🡪 position)

Console.ReadLine()

# Objective 4: Understand how to use selection statements

In this objective you learn how to make different parts of your program run depending on the value of variables.

## Tasks

1. Try entering the following commands and see what happens:

'Using selection statements to check variables

'Declare a number variable

Dim number As Integer

'Ask user for the number

Console.WriteLine("Enter a number 1-3: ")

number = Console.ReadLine

'Check the value of the number

If number = 1 Then Console.WriteLine("You entered one")

If number = 2 Then Console.WriteLine("You entered two")

If number = 3 Then Console.WriteLine("You entered three")

Console.ReadLine()

1. Try entering the following commands and run the program three times with the test data:  
   6, 51 and 70.

'Works out whether a candidate passed or failed

Dim score As Integer

Console.WriteLine("Enter the test score 1-100: ")

score = Console.ReadLine

'Branch depending on the value of the variable

If score > 40 Then

Console.WriteLine("You passed")

Else

Console.WriteLine("You failed")

End If

Note how the program only executes instructions if the condition is true i.e. the score is more than 40, and how it is possible to use an else statement to run an alternative set of commands if the condition is false.

1. Change the program so that you have to score over 50 to pass.
2. Try entering the following commands and run the program three times with the test data:  
   6 & 4, 51 & 51.

'Returns whether two numbers are the same

Dim num1, num2 As Integer

Console.Write("Enter the first number: ")

num1 = Console.ReadLine

Console.Write("Enter the second number: ")

num2 = Console.ReadLine

'Branch depending on the value of the variable

If num1 <> num2 Then

Console.WriteLine("The numbers are different")

Else

Console.WriteLine("The numbers are the same")

End If

Console.ReadLine()

Note <> means does not equal.

1. Try entering the following commands and run the program three times with the test data:  
   1, 3, 5.

'Using logic operators

Dim choice As String

Console.WriteLine("Enter a number 1-3: ")

choice = Console.ReadLine

If choice > "0" And choice < "3" Then

Console.WriteLine("Valid input")

Else

Console.WriteLine("Invalid input")

End If

Console.ReadLine()

Note how ‘And’ can be used to check if both conditions are true. You can also use ‘Or’ if only one condition needs to be true and ‘Not’ as an alternative to does not equal.

Note how the number was input as a string, not an integer, yet greater than and less than operators still work to check if the input was between 1 and 2. That’s because the ASCII value of the character is being checked, not the actual number. This approach is helpful as it prevents the program crashing if a character is input instead of a number.

1. Try entering the following commands and see what happens:

'Using case selection

Dim choice As String

'Ask user for the number

Console.WriteLine("1. Add numbers")

Console.WriteLine("2. Subtract numbers")

Console.WriteLine("3. Quit")

Console.WriteLine("Enter your choice: ")

choice = Console.ReadLine

'Multiple branches depending on selection

Select Case choice

Case Is = "1"

Console.WriteLine("Add numbers chosen")

Case Is = "2"

Console.WriteLine("Subtract numbers chosen")

Case Is = "3"

Console.WriteLine("Quit chosen")

End Select

Console.ReadLine()

Note how it is possible to use a Select Case… End Select structure to have multiple branches rather than just true or false.

# Objective 4: Key learning points How to use selection statements

* Checking the value of a variable and executing instructions depending on the outcome of the check is known as a **selection**.
* The instructions that are executed as a result of a selection is known as a **program branch**.
* The logical checking of a variable against another value is known as a **condition**.
* Select/Case commands can be used instead of multiple if statements.
* Code for each program branch should be indented.
* It is good practice to comment a selection to explain its purpose.
* One ‘If’ can be placed inside another ‘If’, this is known as **nesting**.

# Objective 4: Key words

### If… then… else… endif

Example code: If (x>0 And x<5) or x=10 then  
 …  
Else  
 …  
End If

x is the variable being checked. Multiple variables can be checked in one condition. Use brackets to express order of priority. In the example code, x must be between 0 and 5 or equal to 10 for the condition to be true. Else is optional and is used to branch if the condition is false.

Logical operators that can be used in conditions include:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| = | equals | < | less than | And | both conditions must be true |
| <> | does not equal | <= | less than or equal to | Or | one condition must be true |
|  |  | > | greater than | Not | condition is not true |
|  |  | >= | greater than or equal to |  |  |

### Select… Case… End Select

Example code: Select Case x  
 Case Is = 3  
 …  
 Case Is < 3  
 …  
 Case 10 to 20  
 …  
 Case 20, 23, 28  
 …  
 Case Else  
 …  
 End Select

x is the variable being checked. Select Case is used as an alternative to multiple if statements.

# Objective 5: Understand how to use arithmetic operations, random numbers and type conversions

In this objective you learn how to perform mathematical calculations using power, modulus and integer division. Random numbers allow for some unpredictability which is useful for games. You also learn how to convert from one data type to another.

## Tasks

1. Try entering the following program to see how arithmetic operators work:

'Declare variables

Dim power\_of\_result As Integer

Dim number1, number2 As Integer

Dim division\_result As Decimal

Dim integer\_division\_result As Integer

Dim modulus\_result As Integer

'Get user input

Console.Write("Enter first number: ")

number1 = Console.ReadLine

Console.Write("Enter second number: ")

number2 = Console.ReadLine

'Make calculations

power\_of\_result = number1 ^ number2

division\_result = number1 / number2

integer\_division\_result = number1 \ number2

modulus\_result = number1 Mod number2

'Output results

Console.WriteLine()

Console.WriteLine(number1 & " to the power of " & number2 & 🡪  
" is " & power\_of\_result)

Console.WriteLine(number1 & " divided by " & number2 & 🡪  
" is " & division\_result)

Console.WriteLine(number1 & " divided by " & number2 & 🡪  
" is " & integer\_division\_result)

Console.WriteLine(number1 & " divided by " & number2 & 🡪  
" has a remainder of " & modulus\_result)

Console.ReadLine()

End Sub

1. Try entering the following commands and see what happens:

'Declare variables

Dim random\_number As Integer

'Generate a new random number seed

Randomize()

'Roll the dice

random\_number = CInt(Rnd() \* 5) + 1

Console.WriteLine("You rolled a " & random\_number)

Console.ReadLine()

1. Change the program so that it outputs a 10 sided dice.
2. Change the program so the user can choose to roll a 4, 6 or 12 sided dice.

# Objective 5: Key learning points How to use arithmetic operations and random numbers

* / and \ both perform division. \ gives an integer answer.
* **Modulus** is the remainder of a division. E.g. 8 divided by 5 is one with 3 left over (remainder)
* Commands can be put inside commands. E.g. CInt(Rnd()) is two commands, one inside the other. This example is the integer of a random number.
* A variable is stored with a specific data type that determines how much memory it uses and how it is handled in the program. Typical data types include: integer, real/float (decimal), character, string, date and boolean.
* Some data types have identifying characters. E.g. a string is enclosed in quotes.
* The integer 7, is therefore not the same as the character “7”. They have different binary values.
* It is possible to change a variable’s data type using conversion functions.

Mathematical operators that can be used in conditions include:

|  |  |  |  |
| --- | --- | --- | --- |
| + | addition | ^ | power of |
| - | subtraction | \ | integer division |
| \* | multiplication | mod | modulus (remainder after division) |
| / | division |  |  |

# Objective 5: Key words

### Mod

Example code: x = y mod z

x becomes the remainder of y divided by z. This is useful to determine if one number is exactly divisible by another. For example with x = 2000 mod 4, x would be 0 because 2000 is exactly divisible by 4.

### Rnd()

Example code: x = CInt(y \* rnd()) +1

x becomes a random number between 0 and y. Rnd returns a decimal number between 0 and 1. We multiply this by the range of numbers and convert it to an integer in order to get a single round number. +1 is optional. By adding 1 the code will not return 0 as a random number. If that is desired, do not include the +1, but remember y is the range of numbers, so if y is 5 then the returned number will be between 0 and 4.

### Randomize()

Example code: Randomize()

Computers are not able to generate truly random numbers. Instead they make a calculation using “seed” numbers, and return numbers from a sequence. Without the keyword Randomize you would get the same set of random numbers each time the program is run! Randomize creates a new seed and therefore a new set of random numbers each time the program is run.

### CInt()

Example code: x = CInt(y)

x becomes the integer of y, or y rounded down to the nearest whole number.

### CDec()

Example code: x = CDec(y)

x becomes the decimal value of y

### CStr()

Example code: x = CStr(y)

x becomes a string of y

# Objective 6: Understand counter controlled iterations

In this objective you learn how to get code to repeat itself a given number of times without having to enter lots of repeating statements.

## Tasks

1. Try entering the following commands and see what happens:

For counter = 1 To 5

Console.WriteLine("This is how you get code to repeat")

Next

Console.ReadLine()

1. Try changing the number 5 and see what happens. Also try changing the number 1 and see what happens.
2. Modify the code as shown below. Run the program and see what happens. Note how the value of the counter variable changes.

For counter = 7 To 12

Console.WriteLine("The counter is " & counter)

Next

Console.ReadLine()

1. Enter the following commands and see what happens:

Dim word As String = "Hello"

For counter = 1 To Len(word)

Console.WriteLine("Letter " & counter & " is " & 🡪  
Mid(word, counter, 1))

Next

Console.ReadLine()

Note how we can now loop through all the letters in a word by combining an iteration with string manipulation commands. Also note how commands inside an iteration are indented to make the block of code easier to read.

# Objective 6: Key learning points Counter controlled iterations

* An **iteration** is a section of code that is repeated.
* Iterations are also known as **loops**.
* Counter controlled iterations are used when you want to iterate a known number of times.
* Counter controlled iterations use a variable which can be used within the code to know how many times the code has been run.
* Code is **indented** within an iteration to make it easier to read.
* It would be good practice to comment code before an iteration to explain what is happening.
* **Incrementing** a variable increases its value by one.
* **Decrementing** a variable decreases its value by one.
* One ‘For’ loop can be put inside another ‘For’ loop. This is known as **nesting**.

# Objective 6: Key words

### For… Next

Example code: For counter = 1 to 5 step 1  
 Console.WriteLine(counter)  
Next

Executes code within the ‘For/Next’ structure a known number of times. Counter is the variable that is counting how many times the code has iterated. 1 is the starting value. 5 is the stopping value.

Step determines by how much the counter increments. By default this is 1 and therefore step 1 is not necessary. To count backwards, a negative value can be used for step. E.g.

For counter = 5 to 1 step -1  
 Console.WriteLine(counter)  
Next

# Objective 7: Understand condition controlled iterations

In this objective you learn how to get code to repeat continually until a condition is met.

## Tasks

1. Try entering the following program and see what happens by entering a few numbers and ending by entering zero:

'Program to add up numbers until 0 is entered

'Initialise variables

Dim total As Decimal = 0

Dim number\_entered As Decimal = 1

'Enter numbers to add

Do While number\_entered > 0

Console.WriteLine("Enter a number")

number\_entered = Console.ReadLine

'Add number to total

total = total + number\_entered

Loop

'Output result

Console.WriteLine("The sum of the numbers is: " & total)

Console.ReadLine()

1. Compare the program above to this code written slightly differently:

'Program to add up numbers until 0 is entered

'Initialise variables

Dim total As Decimal = 0

Dim number\_entered As Decimal

'Enter numbers to add

Do

Console.WriteLine("Enter a number")

number\_entered = Console.ReadLine

'Add number to total

total = total + number\_entered

Loop While number\_entered > 0

'Output result

Console.WriteLine("The sum of the numbers is: " & total)

Console.ReadLine()

Note that the While statement can be placed either at the beginning of the iteration (as in program 1) or at the end of the program (as in program 2). Although these iterations both use the same commands, they are not the same.

When the While number\_entered > 0 condition is placed with the Do statement, the code inside the iteration may not execute at all because the value of number\_entered is checked first.

When the While number\_entered > 0 condition is placed with the Loop statement, the code inside the iteration must executae at least once because the value of number\_entered is not checked until the end.

# Objective 7: Key learning points Condition controlled iterations

* When it is not known in advance how many times code needs to be repeated, a condition controlled loop is used.
* Conditions can be set at the end of the iteration so code executes at least once, or at the beginning of the iteration where code inside the iteration may not execute at all.
* Condition controlled iterations are useful when you want to validate data entry by only continuing to the next statements once the data input is correct.
* There are many examples of condition controlled iterations, Do..Loop While and Do While… Loop are just two examples.
* Code should always be indented inside iteration statements.

# Objective 7: Key words

### Do While… Loop

Example code: Do While counter < 3  
 …  
Loop

Checks the condition and if true executes the code before the Loop, returning to check again afterwards.

### Do… Loop While

Example code: Do  
 …  
Loop While counter < 3

Executes the code after the Do statement at least once before checking the condition. If the condition is true the code after to Do statement executes again.

# Objective 8: Understand subroutines, procedures and functions

In this objective you learn how to divide your program into more manageable sections to make it easier to read and reduce the need for entering repeating code.

## Tasks

1. Try entering the following program and see what happens:

'Global variables accessed by all subroutines

Dim MaxNoOfStars, NoOfStars, NoOfSpaces As Integer

'Main program starts here

Sub Main()

InitialiseNoOfSpacesAndStars()

Do

OutputLeadingSpaces()

OutputLineOfStars()

AdjustNoOfSpacesAndStars()

Loop While NoOfStars <= MaxNoOfStars

Console.ReadLine()

End Sub

'Set the size of the pyramid of stars

Sub InitialiseNoOfSpacesAndStars()

Console.Write("How many stars at the base (1-40)? ")

MaxNoOfStars = Console.ReadLine()

'Calculate spaces needed from tip

NoOfSpaces = MaxNoOfStars / 2

'Set tip of pyramid to one star

NoOfStars = 1

End Sub

'Outputs spaces before stars

Sub OutputLeadingSpaces()

Dim count As Integer

For count = 1 To NoOfSpaces Step 1

Console.Write(" ")

Next count

End Sub

'Outputs stars

Sub OutputLineOfStars()

Dim count As Integer

For count = 1 To NoOfStars

Console.Write("\*")

Next

'Move to next line

Console.WriteLine()

End Sub

'Adjusts number of stars & spaces after output

Sub AdjustNoOfSpacesAndStars()

NoOfSpaces = NoOfSpaces - 1

NoOfStars = NoOfStars + 2

End Sub

1. Try entering the following program and see what happens:

'Program to output a set of random numbers

Sub main()

Dim number, max As Integer

Console.Write("How many numbers do you want to output? ")

number = Console.ReadLine()

Console.Write("What is the maximum number? ")

max = Console.ReadLine

outputrandoms(number, max)

Console.ReadLine()

End Sub

'Output random numbers

Sub outputrandoms(ByVal n As Integer, ByVal m As Integer)

Dim counter, randomnum As Integer

For counter = 1 To n

randomnum = CInt(Rnd() \* m)

Console.WriteLine("Number " & counter & " is " & randomnum)

Next

End Sub

There are two ways to share data between subroutines. In the first program, the variables were declared outside of any subroutine. That means they are global variables to the program, and their values can be accessed, shared and changed by any subroutine in the program.

In the second program the variables ‘number’ and ‘max’ are declared in subroutine main(). Their values are then passed into the subroutine outputrandoms. The values of the variables become n and m in the subroutine outputrandoms. The variables n and m are different variables to number and max, but the value stored in number and max was passed into n and m. This is also known as passing by value, hence the command ByVal. In this example, the variables n and m are lost at the end of the outputrandoms subroutine. That means they are local variables to the subroutine.

1. Try entering this program and see what happens:

'How functions can be used

Sub main()

Dim num1, num2 As Integer

Console.WriteLine("Enter the first number: ")

num1 = Console.ReadLine

Console.WriteLine("Enter the second number: ")

num2 = Console.ReadLine

Console.WriteLine("The floor number is: " & floor(num1, num2))

Console.ReadLine()

End Sub

'Returns a positive number from subtraction

Function floor(ByVal a As Integer, ByVal b As Integer) As Integer

Dim f As Integer

f = a - b

If f > 0 Then Return f Else Return 0

End Function

1. Change the function so it receives the length of two sides of a right angled triangle, and returns the length of the hypotenuse (h2 = o2+a2)

# Objective 8: Key learning points Understand subroutines, procedures and functions

* Subroutines are **sections of code** to break a longer programs into smaller pieces. Thereby making them easier to read and more manageable for teams of programmers to work together on one program.
* Subroutines are also known as **procedures**.
* **Functions** carry out small tasks on data by taking one or more **parameters** and **returning** a result.
* Subroutines/procedures may also take parameters, but do not return a result.
* It is good practice to comment each subroutine or function to explain its purpose.
* Variables declared outside of all subroutines are known as **global variables**. They are available to all subroutines and functions.
* Global variables are not memory efficient since they hold memory for the entire time a program is running. They should therefore be kept to a minimum.
* Variables declared inside a subroutine or function are known as **local variables**. They lose their value when the routine ends, releasing memory.
* Passing variables between subroutines is known as **parameter passing**.

# Objective 8: Key words

### Sub… End Sub

Example code: Sub max (ByVal x as integer, ByVal y as integer)  
   
End Sub

Creates a subroutine called max. Two parameters are passed into max, with local variables x and y holding their values.

max would be called with: x=max(6,9)  
6 would be passed as a parameter to variable x. 9 would be passed as a parameter to variable y.

### Function… Return… End Function

Example code: Function max (ByVal x as integer, ByVal y as integer) As Decimal

Return z  
End Sub

Creates a function called max. Two parameters are passed into max, with local variables x and y holding their values.

max would be called with: n = max(6,9)  
6 would be passed as a parameter to variable x. 9 would be passed as a parameter to variable y.  
The value returned to n would be z, which in this case is a decimal because the function was declared with As Decimal to indicate the data type of the return value.

# Objective 9: Understand arrays and lists

In this objective you learn how to store multiple items without having lots of different variables.

## Tasks

1. Try entering the following program and see what happens:

'Declare array and data set

Dim sentence() As String = {"The", "quick", "grey", "fox", "jumps"}

'Output contents of array

Console.WriteLine(sentence(0))

Console.WriteLine(sentence(1))

Console.WriteLine(sentence(2))

Console.WriteLine(sentence(3))

Console.WriteLine(sentence(4))

Console.ReadLine()

Sentence is an example of an array: a group of elements with the same name and data type.

Think of an array as a table:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Identifier: | sentence | | | | |
| Index: | 0 | 1 | 2 | 3 | 4 |
| Data: | The | quick | grey | fox | jumps |

1. Try this modification to the program to allow the user to change the colour of the fox:

'Declare array and data set

Dim sentence() As String = {"The", "quick", "grey", "fox", "jumps"}

Console.WriteLine("Enter new colour: ")

sentence(2) = Console.ReadLine()

Console.WriteLine(sentence(0))

Console.WriteLine(sentence(1))

Console.WriteLine(sentence(2))

Console.WriteLine(sentence(3))

Console.WriteLine(sentence(4))

Console.ReadLine()

Note how it is possible to change an individual element of an array by referring to the index.

1. You don’t always want to populate the array with data when it is declared. For example, to declare an array of 100 elements, you would use the alternative syntax:

Dim item(99) As String

You can then put data into the array using the syntax:

item(0) = "Alpha"

item(99) = "Omega"

Note that indexes always start at zero. So, one hundred elements is 0-99.

1. The power of arrays often comes by combining them with iterations.  
   Enter this code as an alternative to the first program and notice how the amount of code is much reduced when you use a for loop to output each of the elements in the array.

'Declare array and data set

Dim sentence() As String = {"The", "quick", "grey", "fox", "jumps"}

'Use a loop to cycle through the indexes

For counter = 0 To 4

'Output the element

Console.WriteLine(sentence(counter))

Next

Console.ReadLine()

1. Try entering this program and test by entering 8743 when prompted for a product code:

'Declare array and set data

Dim product() As String = {"1262", "Cornflakes", "£1.40", "8743", 🡪 "Weetabix", "£1.20", "9512", "Rice Krispies", "£1.32"}

Dim product\_code As String 'Product code to find

Dim counter As Integer 'Used by iteration

Dim found As Boolean 'Whether product was found in array

'Ask use for the product code to find

Console.WriteLine("Enter the product to find: ")

product\_code = Console.ReadLine

'Use a loop to cycle through the indexes

counter = 0

found = False

Do

'Output the product if found

If product(counter) = product\_code Then

Console.WriteLine(product(counter + 1))

Console.WriteLine(product(counter + 2))

found = True

End If

counter = counter + 1

Loop While found = False And counter <> 9

'Output message if product is not found

If counter = 9 Then Console.WriteLine("Product not found")

Console.ReadLine()

Note how an iteration and Boolean variable can be used to go through all the indexes of an array to find an item.

This is not the best way to write this program, but does illustrate how a number of programming concepts can be combined.

An alternative to an array is a list. An array is thought of as a static data structure because it has a fixed size at run time. A list is a dynamic data structure that grows and shrinks to accommodate the data stored in it. Lists also contain additional methods and properties that allow you to interrogate or manipulate the list more easily than an array.

1. Try entering the following program and see what happens:

'Create a new list

Dim inventory As New ArrayList

'Add items to the list

inventory.Add("torch")

inventory.Add("gold coin")

inventory.Add("key")

'Remove items from the list

inventory.Remove("gold coin")

'Sort the items into alphabetical order

inventory.Sort()

'Output all the items in the list

For counter = 0 To inventory.Count - 1

Console.WriteLine(inventory.Item(counter))

Next

1. Change the program so that a ‘pick axe’ is added to the inventory.
2. Change the program so the inventory is output and the user can choose which item to drop.

# Objective 9: Key learning points Understand arrays and lists

* Arrays are variables that have multiple elements, **all of the same data type**.
* Arrays are a **static data structure** because the size of the array does not change when the program is running.
* Lists are **dynamic data structures** because the size of the list changes when the program is running.
* An array or list has an **identifier**: the name of the array.
* An array or list has an **index** which points to one element of the array/list.
* Indexes start at 0.
* An array has multiple **elements**: the data stored in it.
* Arrays can be single dimension (one set of data) or be multi-dimension (having more than one set of data)
* Lists have **methods** that are operations that can be performed on the list with no additional code written by the programmer.

# Objective 9: Key words

### Dim

Example code: Dim x(5) as integer

Creates a single dimension array called x with 6 elements (indexes 0-5).

x(4) = 8 would put the number 8 into the array x at index position 4.

Data can be populated into an array using square brackets and commas to separate items:

x = {4, 7, 3, 1, 9}

Multi-dimensional arrays are declared with maximum indexes separated with commas e.g. Dim x(10,2)  
This can be visualised as a table with ten rows and two columns.

x(4,2) = 8 would put the number 8 into the array at row 4, column 2.

y = x(6,4) would put the value in row 6, column 4 into variable y.

### ArrayList

Example code: Dim inventory as New ArrayList

Creates a new list called inventory.

### *list*.Add

Example code: *listname*.Add(item)

Adds item to the list.

### *list*.Insert

Example code: *listname*.Insert(index,item)

Inserts item at position index.

### *list*.Remove

Example code: *listname*.Remove(item)

Removes item from the list.

### *list*.RemoveAt

Example code: *listname*.RemoveAt(index)

Removes an item from the list, stored at the index. Indexes start at 0 with the first item.

### *list*.Count

Example code: *listname*.Count

Outputs the number of items in the list.

### *list*.Item

Example code: *listname*.Item(index) = x

x = *listname*.Item(index)

Sets or gets an item from a list at an index.

### *list*.Sort

Example code: *listname*.Sort()

Sorts all the items in a list.

### *list*.Clear

Example code: *listname*.Clear()

Removes all the items from a list.

### *list*.Contains

Example code: if *listname*.Contains(x) then

Returns whether the list contains item x.

# Objective 10: Understand serial files

In this objective you learn how to read and write data to files so it can be saved and loaded again later.

## Tasks

1. Try entering the following program and see what happens:

**YOU WILL NEED TO CHANGE m:\ TO A PREFERRED FOLDER**

'Open a text file for writing

Dim file As String = "m:\datafile.txt"

FileOpen(1, file, OpenMode.Output)

'Write data to the file

PrintLine(1, "This is a simple way to save data")

PrintLine(1, "one line at a time")

'Empty buffer & close the file

FileClose(1)

Console.WriteLine("File created.")

Console.ReadLine()

Open the file test.txt and look at its contents.

Note the file variable contains the path and the filename. You will need to change this so the file is written to your My Documents folder.

Note how OpenMode.Output specifies that we are writing to the file. Other modes include Input for reading, Append for adding to the end of an existing file and Random for accessing any part of a file.

1. Try entering the following program and see what happens:

'Open a text file for reading

Dim file As String = "m:\datafile.txt"

Dim data As String

FileOpen(1, file, OpenMode.Input)

'Read data from the file

data = LineInput(1)

Console.WriteLine(data)

data = LineInput(1)

Console.WriteLine(data)

'Close the file

FileClose(1)

Console.WriteLine("File closed.")

Console.ReadLine()

1. Change the program so that 3 lines of text are written to a file and read back in again.

It is often useful to read in all the data from a file line by line until you reach the end of the file. To do this we would want to combine an iteration with a file read.

1. Try entering the following program and see what happens:

'Open a text file for reading

Dim file As String = "m:\datafile.txt"

Dim data As String

FileOpen(1, file, OpenMode.Input)

'Check if End Of File

Do While Not EOF(1)

'Read data from the file

data = LineInput(1)

Console.WriteLine(data)

Loop

'Close the file

FileClose(1)

Console.WriteLine("File closed.")

Console.ReadLine()

# Objective 10: Key learning points Understand serial files

* Serial files store data with no order to the data maintained.
* To search data from a serial file you begin at the start of the file and read all the data until the item is found.
* Data cannot be deleted from a serial file without creating a new file and copying all the data except the item you want to delete.
* Data cannot be changed in a serial file without creating a new file, copying all the data across to a new file, inserting the change at the appropriate point.
* Data can be appended to a serial file.
* A file can be open for reading or writing, but not reading and writing at the same time.
* Serial files are quite limiting, but are useful for simple data sets and configuration files. Other types of files include: sequential files where order of the data is maintained, index sequential files for large data sets and random files which allow you to access any item without searching through the file from the start.

# Objective 10: Key words

### FileOpen()

Example code: FileOpen(1,"*filename*",OpenMode.Input)

Opens a file specified by the path and filename.

1 refers to a file number. Many files can be open at once and are each referred to by their file number.

OpenMode.Input - for reading data  
OpenMode.Output - for writing data, over-writing a file that may already exist  
OpenMode.Append - for adding data to the end of the file without overwriting existing contents

### LineInput()

Example code: x = LineInput(1)

Reads a single line of data from an open file into x.

### PrintLine()

Example code: PrintLine(1,"data to write")

Writes a single item of data to an open file.

### CloseFile()

Example code: CloseFile(1)

Commits any data in the file buffer to storage and closes the file. If a file is not closed, data will not be written.

### EOF()

Example code: Do While Not EOF(1)

Used to check the end of file marker in the file to determine if the end of file has been reached.

# Objective 11: How to handle exceptions for validation

In this objective you learn how to prevent your program crashing if it receives invalid data.

## Tasks

1. Try entering the following program and use the test table of data to see what happens:

'Run time errors

Dim num1, num2, answer As Decimal

Console.WriteLine("Enter a number:")

num1 = Console.ReadLine

Console.WriteLine("Enter a number:")

num2 = Console.ReadLine

answer = num1 / num2

Console.WriteLine(num1 & " divided by " & num2 & " is " & answer)

Console.ReadLine()

|  |  |  |  |
| --- | --- | --- | --- |
| **Test** | **First number** | **Second number** | **Expected** |
| 1 | 6 | 2 | 3.0 |
| 2 | 8 | 7 | 1.142 |
| 3 | 4 | 0 | Crash – divide by zero |
| 4 | 5 | b | Crash – invalid character entered |

A program should not crash because it has bad data input. The potential for errors should be considered by the programmer.

1. Try entering the following program and use the test table of data to see what happens:

'Run time errors

Dim num1, num2, answer As Decimal

'Get numbers

Try

Console.WriteLine("Enter a number:")

num1 = Console.ReadLine

Console.WriteLine("Enter a number:")

num2 = Console.ReadLine

Catch

Console.WriteLine("Invalid entry.")

End Try

'Output

Try

answer = num1 / num2

Console.WriteLine(num1 & " divided by " & num2 & " is " & answer)

Catch

Console.WriteLine("Unable to divide.")

End Try

Console.ReadLine()

# Objective 11: Key learning points How to handle exceptions for validation

* There are 3 types of errors in programming:
  + **Syntax errors** when you mistype a keyword, the program doesn’t recognise it as a valid command and will not run.
  + **Logic errors** where the program runs but you get the wrong result, perhaps not always consistently! Logic errors are called **bugs**.
  + **Run-time errors** that occur in exceptional circumstances such as division by zero and incorrect data type input. These should be trapped with **exception handling** commands.
* Fixing errors in code is known as **debugging**.
* Preventing invalid data input is known as **validation**. There are a number of different types of validation including:
  + Presence checks: did the user actually enter any data?
  + Range checks: does the input fall within a valid range of numbers?
  + Format check: does the input match a recognised format, e.g. LLNN NLL for postcode
  + Length check: is the input the required length of characters, e.g. minimum 8 character password.
* Most validation checks can be performed with selection and iteration statements, but sometimes an error could occur if one data type is converted to another, e.g. a string to an integer if the character is not a number.
* The development environment may contain a variety of tools to assist you in debugging a program:
  + Highlighting incorrect key words.
  + Compiler output errors that tell you why your program won’t run.
  + **Break points** to stop the program at a particular point to check the value of variables.
  + **Stepping** through the program line by line to see what command is executing.
  + **Tracing** the value of variables as they step through the program.
  + An immediate window where you can try commands and see the output.

# Objective 11: Key words

### Try… Catch… End Try

Example code: Try  
x = y/z  
  
Catch  
Console.WriteLine("Division by zero")  
  
End Try

‘Try’ is the keyword to put before a run time error is likely to occur. The code following ‘Catch is the code to execute if an error occurs.

**Now for the challenges….**

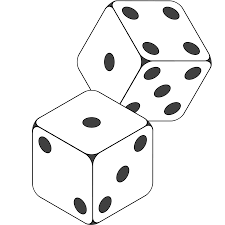
# Objective 1: Challenges

You should attempt both challenges.

## Visual dice challenge

Difficulty: 🛠

Write a program that will output the number 5 on a dice like this:

**oooooooooooo**

**o o**

**o # # o**

**o # o**

**o # # o**

**o o**

**oooooooooooo**

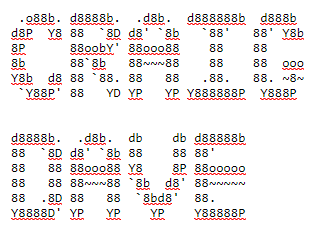
## ASCII art challenge

Difficulty: 🛠

ASCII *(pronounced: as-key)* art is a graphic design technique of making pictures using only the 95 printable characters on a keyboard.

Write a program that outputs your name in ASCII Art.

You may find it easier to create the final output in Notepad first and then copy the lines into the programming language.



# Objective 2: Challenges

You should attempt one difficulty 🛠 challenge, one difficulty 🛠🛠 challenge, and one difficulty 🛠🛠🛠 challenge.

## Simple adder challenge

Difficulty: 🛠

Write a program that asks the user for two numbers, adds them together and outputs for example:  
You entered numbers 5 and 12  
They add up to 17

## Test marks challenge

Difficulty: 🛠

Write a program that will ask the user to enter three test marks out of 100 and output the average.

## Temperature converter challenge

Difficulty: 🛠

Write a program to enter a temperature in degrees Fahrenheit and display the equivalent temperature in degrees Centigrade.  
The formula for conversion is Centigrade = (Fahrenheit – 32) \* (5/9)

## Height & weight challenge

Difficulty: 🛠🛠

Write a program to convert a person’s height in inches into centimetres and their weight in stones into kilograms. (1 inch = 2.54 cm and 1 stone = 6.364 kg)

## Toy cars challenge

Difficulty: 🛠🛠

A worker gets paid £12/hour plus £0.60 for every toy car they make in a factory. Write a program that allows the worker to enter the number of hours they have worked and the number of cars they have made. The program should output their wages for the day.

## Fish tank volume challenge

Difficulty: 🛠🛠

Write a program that will ask the user to enter the length, depth and height of a fish tank in cm. Calculate the volume of water required to fill the tank and display this volume in litres and UK gallons.  
  
To calculate volume in litres, multiply length by depth by height and divide by 1000.

## Circle properties challenge

Difficulty: 🛠🛠🛠

Consider this circle:  
  
  
  
  
  
  
  
  
  
  
  
Write a program that:

arc angle

sector

diameter

circumference

arc length

radius

* Asks the user to enter the diameter of a circle.
* Asks the user to enter the arc angle.
* Outputs the radius of the circle (diameter divided by 2).
* Outputs the area of the circle (3.14 multiplied by the radius squared).
* Outputs the circumference of the circle (3.14 multiplied by the diameter).
* Outputs the arc length (circumference multiplied by the arc angle, divided by 360).

# Objective 3: Challenges

You should attempt one difficulty 🛠 challenge, one difficulty 🛠🛠 challenge, and one difficulty 🛠🛠🛠 challenge.

## Initials & surname challenge

Difficulty: 🛠

Write the program from the flowchart below to ask for a person’s forename and surname, outputting their first initial and the rest of their name in capitals. E.g. ben white = B WHITE

Start

Input forename

Input surname

Set initial = to first letter of forename

Set upperinitial = to uppercase of initial

Set uppersurname = to uppercase of surname

Output upperinitial and uppersurname

Stop

## Airline ticket challenge

Difficulty: 🛠🛠

Write a program that allows the user to input the name of two cities. The program should then output the first 4 characters of each city in capital letters, separated by a dash. For example, London & Madrid would be: LOND-MADR

## Name separator challenge

Difficulty: 🛠🛠🛠

Write a program that allows the user to enter their full name on one line. The program should then output for example:  
  
Forename: Simon  
Surname: Hall

The program needs to find the space and then extract the characters to the left and right of the space into two different variables.

# Objective 4: Challenges

You should attempt one difficulty 🛠 challenge, one difficulty 🛠🛠 challenge, and one difficulty 🛠🛠🛠 challenge.

## Under age challenge

Difficulty: 🛠

Write a program that asks for your age. If you are over 18 it outputs the message, “Over 18”, otherwise it outputs, “Under age”.

## Water temperature challenge

Difficulty: 🛠

Write a program that reads in the temperature of water in a container in Centigrade and displays a message stating whether the water is frozen (zero or below), boiling (100 or greater) or neither.

## Vocational grade challenge

Difficulty: 🛠

Write a program that allows you to enter a test mark out of 100.  
The program outputs “FAIL” for a score less than 40, “PASS” for a score of 40 or more, “MERIT” for a score of 60 or more and “DISTINCTION” for a score of 80 or more.

## Extended visual dice challenge

Difficulty: 🛠

For a six sided dice, write a program that asks for a number and outputs that number as a graphical dice. E.g.

**oooooooooooo**

**o o**

**o # o**

**o # o**

**o # o**

**o o**

**oooooooooooo**

## Greatest number challenge

Difficulty: 🛠

Write a program to display the larger of two numbers entered:

Start

Is number1 greater than number2?

Stop

Yes

No

Input number1 from keyboard

Input number2 from keyboard

Output, “The largest number is “ number1

Output, “The largest number is “ number2

## Nitrate challenge

Difficulty: 🛠🛠

When keeping fish, one of the goals to reduce algae is to keep nitrates to a minimum. One way of doing this is to dose a carbon source which nitrifying bacteria within an aquarium consume together with nitrates. The carbon source has to be dosed very precisely. Write this program to determine the dose:

Start

Is nitrate above 10?

Is nitrate above 2.5?

Stop

Yes

No

Yes

No

Enter the nitrate level: a number  
between 1 and 50

Is nitrate above 1?

Output “Dose 3ml”

Output “Dose 2ml”

Yes

Output “Dose 0.5ml”

No

Output “Dose 1ml”

## Portfolio grade challenge

Difficulty: 🛠🛠

Write a program that inputs a mark from the keyboard for sections of a project: ‘analysis’, ‘design’, ‘implementation’ and ‘evaluation’. The program should output the total mark, the grade, and how many more marks were needed to get into the next mark band.

Grades are:

<2 U

2 1

4 2

13 3

22 4

31 5

41 6

54 7

67 8

80 9

## Periodic table challenge

Difficulty: 🛠🛠🛠

Write a program that asks the user to enter the symbol or name of an element, or group it belongs to. The program should output the name of the element and its atomic weight. E.g.

The user enters Li. The program outputs:

Element: Lithium (Li)  
Atomic weight: 6.94  
Group: Alkali metals

If the user enters Alkali metals, the program outputs the data for all the elements in the alkali metals group.

You only need to get this working for 6 elements from two different groups.

# Objective 5: Challenges

You should attempt one difficulty 🛠 challenge, one difficulty 🛠🛠 challenge, and one difficulty 🛠🛠🛠 challenge.

## RPG dice challenge

Difficulty: 🛠

In many RPG games, a number of different sided dice are used, not just the standard 6 sided dice. Write a program that asks the user how many sides a dice has and outputs a random number for that dice.

## Divisible challenge

Difficulty: 🛠🛠

Write a program that asks the user for two numbers. The program should output whether the two numbers are exactly divisible by each other. If not, it should return the remainder. If the user enters a 0 the program should give an error message. Use the test table below to understand the correct outputs for this program:

|  |  |  |
| --- | --- | --- |
| **Input 1** | **Input 2** | **Expected output** |
| 3 | 9 | 9 is exactly divisible by 3 |
| 5 | 15 | 15 is exactly divisible by 5 |
| 8 | 23 | 23 is not exactly divisible by 8, there is a remainder of 7 |
| 10 | 12 | 12 is not exactly divisible by 10, there is a remainder of 2 |
| 0 | 7 | Error: you cannot divide by 0 |

## Cash machine challenge

Difficulty: 🛠🛠

A cash machine dispenses £10 and £20 notes to a maximum of £250. Write a program that shows the user their balance of £231, asks them how much to withdraw, ensures this is a valid amount without going overdrawn and with the notes available, and outputs the new balance.

## Dice game challenge

Difficulty: 🛠🛠

Write a program to calculate the score in this simple dice game:

Start

Roll 3 dice

Are all three dice equal?

Score =   
total on the 3 dice

Are two of the dice equal?

Score =   
sum on the 2 equal dice

Score = score minus  
the number on the other dice

Score = 0

Stop

Yes

No

Yes

No

Output score

## Month challenge

Difficulty: 🛠🛠🛠

Write a program that lets the user enter a year, and a month number between 1 and 12. The program outputs the month name for that month number and the number of days in the month. The input 3 would therefore display March has 31 days.

Remember February has 28 or 29 days depending on whether it is a leap year. It is a leap year if the year is exactly divisible by 4.

# Objective 6: Challenges

You should attempt one difficulty 🛠 challenge, one difficulty 🛠🛠 challenge, and one difficulty 🛠🛠🛠 challenge.

## Square numbers challenge

Difficulty: 🛠

Write the program to output all the squares of a number between 1 and 20 as described in the flowchart below.

BEGIN

Output a string using counter and squared variables as shown below

Squared =   
counter \* counter

No

Counter start 1  
Is = 20?

END

Yes

The output should be:

1 squared is 1  
2 squared is 4  
3 squared is 9  
4 squared is 16

## 9 green bottles challenge

Difficulty: 🛠

Write a program that prompts the user to enter the number of bottles. The program outputs:

9 green bottles sitting on the wall  
8 green bottles sitting on the wall  
7 green bottles sitting on the wall etc. etc.

Times table challenge  
Difficulty: 🛠

Write a program that asks the user to enter a number between 1 and 12. The program outputs the times table of that number between 1 and 12.

## Fibonacci sequence challenge

Difficulty: 🛠🛠

A number in the Fibonacci sequence is generated by adding the previous two numbers. By starting with 0 and 1, the first 10 numbers will be: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55. Write a program to produce the sequence of 20 numbers.

## Average calculator challenge

Difficulty: 🛠🛠

Write a program that asks the user to enter how many numbers are to be averaged. The user can then enter the numbers. The program outputs the total and the mean.

## FizzBuzz challenge

Difficulty: 🛠🛠🛠

Write a program that outputs the numbers from 1 to 100. But for multiples of three output “Fizz” instead of the number and for the multiples of five output “Buzz”. For numbers which are multiples of both three and five output “FizzBuzz”

## Maths test challenge

Difficulty: 🛠🛠🛠

Write a program that could be used in an infant school to prompt a child with ten simple random 1 digit maths additions, e.g. 4+7= The user enters their answer and the computer tells them if they are correct. At the end of the test the program outputs how many problems the child got right.

# Objective 7: Challenges

You should attempt one difficulty 🛠 challenge, one difficulty 🛠🛠 challenge, and one difficulty 🛠🛠🛠 challenge.

## Menu selection challenge

Difficulty: 🛠

Write a program that presents a simple menu to the user: 1. Play Game 2. Instructions 3. Quit  
The program should ask the user for a number between 1 and 3 and will output “Let’s go” only when a valid entry is made, repeating the menu selection until a valid entry is made.

## Compound interest challenge

Difficulty: 🛠

Write a program that shows how compound interest grows in a bank savings account. The program should ask the user to input their current balance, the interest rate (0.04 would be 4%) and the desired balance to reach. The program should output the new balance each year until the desired balance is reached. Interest is added to the balance each year.

e.g.

£100 starting balance – 4% interest rate (0.04)  
Year 1: New balance = 100 + ( 100\*0.04) = £104  
Year 2: New balance = 104 + (104\*0.04) = £108.16  
Year 3: New balance = 108.16 + (108.16\*0.04) = £112.49

## Guess the number game challenge

Difficulty: 🛠🛠

The computer guesses a number between 1 and 100. The player has to try and guess the number in as few attempts as possible. When the user enters a number they are told if the guess is too high or too low until the number is guessed correctly. The player is told how many guesses they made. Write a program to play the game.

## Gamertag challenge

Difficulty: 🛠🛠

A gamertag is a username on a typical games console that other players can identify the player with. There are some limitations to them, such as, it cannot be longer than 15 characters.

Write a program to input a valid gamertag and keep asking for a new gamertag until a valid entry is made.

|  |
| --- |
| **Pseudocode** |
| 1.0 Write a comment to explain the program checks the length of gamerTags entered. |
| 2.0 Set a variable called valid\_gamertag = true |
| 3.0 Start a condition controlled iteration to check valid\_gamertag |
| 4.0 Output a message on the screen to say gamertags must be less than 15 characters. |
| 5.0 Get a gamertag from the keyboard and put it into a variable called gamertag. |
| 6.0 Using the function len() put the number of characters in gamertag in a variable called gamertag\_length. |
| 7.0 Check if the length of gamertag is more than 15 characters by comparing gamertag\_length to the number 15. |
| 7.1 Output a message to say ‘your gamertag is too long’ if it is of an invalid length. |
| 7.2 Output a message to say ‘gamertag accepted’ if it is of a valid length. |
| 8.0 End iteration |

## Rock, paper, scissors challenge

Difficulty: 🛠🛠🛠

The computer and player choose one of rock, paper, or scissors. The output of the encounter is then displayed with rock beating scissors, scissors beating paper, and paper beating rock. The winner scores 1 point for a win. The score for both players should be output. The game is won when one player reaches 10 points.

## Happy numbers challenge

Difficulty: 🛠🛠🛠

Write a program to output whether a chosen number is happy or sad. A happy number is a number defined by the following process: Starting with any positive integer, replace the number by the sum of the squares of its digits, and repeat the process until the number either equals 1, or it loops endlessly in a cycle which does not include 1. Those numbers for which this process ends in 1 are happy numbers, while those that do not end in 1 are unhappy numbers (or sad numbers). When the algorithm ends in a cycle of repeating numbers, this cycle always includes the number 4.

For example, 19 is happy, as the associated sequence is:

12 + 92 = 82  
82 + 22 = 68  
62 + 82 = 100  
12 + 02 + 02 = 1.

## XP Challenge

Difficulty: 🛠🛠🛠

In many online multiplayer games, players are awarded experience points (known as XP) for completing a variety of objectives in the game. When the player has accumulated enough XP, their rank increases (they rank up) and their XP resets to zero (although extra XP is carried over). Each rank requires more XP than the one before.

Write a program that:

* Allows the user to keep entering XP for a game until it is 2000 or more.
* When the XP reaches 100, ‘Promoted to Corporal’ is output.

|  |
| --- |
| **Pseudocode** |
| 1.0 Set the player’s total XP to be 0 |
| 2.0 Set the player’s rank to be 1 |
| 3.0 Loop while the player’s XP is less than 2000 |
| 3.1 Ask the user to enter the XP earned in the last game |
| 3.2 Add the XP earned from the last game to the total XP |
| 3.3 Check if the XP earned is more than that required to be a Corporal and that their rank is 1 |
| 3.3.1 Tell the player they have been promoted |
| 3.3.2 Set XP back to zero but carry over any XP left |
| 3.3.3 Set player’s rank to be 2 |
| 3.4 Display the total XP |

* Extend the program so that the player ranks up according to the table below:

|  |  |
| --- | --- |
| **XP needed** | **Rank awarded** |
| 0 | Private |
| 100 | Corporal |
| 300 | Sergeant |
| 700 | Staff Sergeant |
| 1500 | Warrant Officer |

# Objective 8: Challenges

You should attempt one difficulty 🛠 challenge, one difficulty 🛠🛠 challenge, and one difficulty 🛠🛠🛠 challenge.

## VAT challenge

Difficulty: 🛠

Write a program that asks the user for a price of an item. Include a function that returns the VAT for the item. This should be output in the main program.

## Conversion challenge

Difficulty: 🛠🛠

Write a program that provides a conversion utility, converting from metric to imperial measurements. The list of possible conversions can be presented to the user in a menu.

Each conversion should be a separate subroutine or function to allow the program to be modular and added to later.

## Darts challenge

Difficulty: 🛠🛠

Darts is a game where the objective is to reach a score of zero starting from 501. A player throws 3 darts and their total from the throw is subtracted from their score. A player has to achieve a score of exactly zero to win the game. A throw that would result in a score below two does not count.

Create a program to allow the user to enter a total from their darts. The program should then subtract this from their current score if the remaining score would be greater than one.  
The player should win the game when their score becomes exactly zero.

|  |
| --- |
| **Pseudocode** |
| 1.0 Set the player’s score to be 501 |
| 2.0 Display the message: New game. Player starts at 501 points. |
| 3.0 Loop while the player’s score does not equal zero |
| 3.1 Ask the user to enter the total from 3 darts |
| 3.2 Check if the player’s score minus the total is greater than 1 |
| 3.2.1 Subtract the total from the player’s score |
| 3.2.2 Display the player’s new score |
| 3.3 Else check if the player’s score minus the total equals 0 |
| 3.3.1 Subtract the total from the player’s score |
| 3.3.2 Display the message: Player 1 wins the game |

## Darts challenge part 2

Difficulty: 🛠🛠🛠

Allow the player to start a new game when they have won.

Create a two player version of the game where players take it in turns to throw darts and have the total subtracted from their own current score. The player that reaches zero first wins.

## Snake eyes challenge

Difficulty: 🛠🛠🛠

Write a program that plays the following game:

* Player 1 starts the game by rolling two dice.
* The total is the value of the two dice. E.g. 3 and 4, total = 7
* If a player rolls a single 1 on either die, they lose the running total, it is reset to 0 and play passes to the other player without them having a choice to gamble or bank.
* If a player rolls a double 1, they lose their banked total and running total, both are reset to 0 and play passes to the other player without them having a choice to gamble or bank.
* If a 1 was not thrown on either die, the player has the option to gamble and roll the dice again or to bank the accumulated score.
* If the player gambles, the total from the two dice is added to a running total and the dice are rolled again, creating a new total.
* If a player banks, the running total is added to the banked score and the running total is reset back to 0.
* If a player banks, his turn is over and play passes to the other player.
* The game is won by the first person reaching a banked total of 100 or more.

# Objective 9: Challenges

You should attempt one difficulty 🛠 challenge, one difficulty 🛠🛠 challenge, and one difficulty 🛠🛠🛠 challenge.

## Text dice challenge

Difficulty: 🛠

Write a program that generates a random number and outputs the word, “one”, “two”, “three”, “four”, “five” or “six” without using a condition statement like ‘if’.

## Notebook challenge

Difficulty: 🛠🛠

Write a program that allows the user to store up to 10 notes. The program should have this functionality on a continual loop:

1. Output all the notes stored, numbered 0-9.
2. Ask the user to enter a number for a note to change.
3. Ask the user to enter the new note.
4. Over-write any existing note stored at that position.

## Currency converter challenge

Difficulty: 🛠🛠

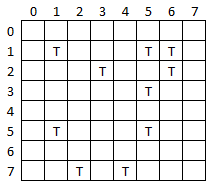
Write a program that stores the names and exchange rates of currencies in an array. The program should ask the user to enter the number of British pounds and a currency to convert into. The program should then output the exchange rate followed by the conversion from British pounds.

## Tanks challenge

Difficulty: 🛠🛠🛠

Tanks is a game for two players. At the start of the game, each player places 10 tanks on a board but does not reveal their location to their opponent. Each tank occupies one square on the board. Each player takes it in turn to enter a grid reference, e.g. 1,5. The player destroys the tank if the square is occupied by a tank. The player to destroy all their opponent’s tanks first wins.

Example tanks board:



# Objective 10: Challenges

You should attempt one difficulty 🛠 challenge, one difficulty 🛠🛠 challenge, and one difficulty 🛠🛠🛠 challenge.

## RPG attributes challenge

Difficulty: 🛠

Write a program that allows the user to enter the attributes for a character in an RPG game. This includes their name, attack and defence capability between 0 and 100. The program should write the data to a file.

Write a second program that reads the data from the file and writes it to the screen.

## Quote of the day challenge

Difficulty: 🛠🛠

Write a program that outputs a random ‘quote of the day’, stored in a text file, created in Notepad.  
e.g. “If you fell down yesterday, stand up today”.

## Product catalogue challenge

Difficulty: 🛠🛠🛠

Create a program that allows the user to:

* Enter a new product to add to the catalogue.
* Output the items in the catalogue.
* Find a product in the catalogue and output the price.

All data must be saved and read from a file.

More Challenges

These challenges should only be attempted after difficulty 🛠🛠🛠 challenges have been successfully completed in all the required objectives.

## Word extractor challenge

Difficulty: 🛠🛠🛠🛠  
Requires knowledge of objectives 1-3

Write a program that outputs the sentence: Quick brown fox jumps over the lazy dog. The user can then enter the word to be cut from the sentence. The sentence is then output with the word removed.

## Hours worked challenge

Difficulty: 🛠🛠🛠🛠  
Requires knowledge of objectives 1-4

Write a program that asks the user for the number of hours worked this week and their hourly rate of pay. The program should calculate the gross pay. If the number of hours worked is greater than 40, the extra hours are paid at 1.5 times the rate. The program should display an error message if the number of hours worked is not in the range 0 to 60.

## ROT13 challenge

Difficulty: 🛠🛠🛠🛠  
Requires knowledge of objectives 1-6

ROT13 is a simple letter substitution cipher that replaces a letter with the letter 13 letters after it in the alphabet. ROT13 is a special case of the Caesar cipher, developed in ancient Rome. ROT13 is used in online forums as a means of hiding spoilers, punchlines and puzzle solutions from the casual glance. Create a program that allows the user to enter plain text, and the ROT13 cipher is output. Extend the program to allow cipher text to be input, with the plain text output.

## Letter game challenge

Difficulty: 🛠🛠🛠🛠  
Requires knowledge of objectives 1-6

A word game awards points for the letters used in a word. The lower the frequency of the letter in the English language, the higher the score for the letter. Write a program that asks the user to input a word. The program should then output the score for the word according to the following rules:

|  |  |  |  |
| --- | --- | --- | --- |
| Letter | Points | Letter | Points |
| E | 1 | M | 14 |
| A | 2 | H | 15 |
| R | 3 | G | 16 |
| I | 4 | B | 17 |
| O | 5 | F | 18 |
| T | 6 | Y | 19 |
| N | 7 | W | 20 |
| S | 8 | K | 21 |
| L | 9 | V | 22 |
| C | 10 | X | 23 |
| U | 11 | Z | 24 |
| D | 12 | J | 25 |
| P | 13 | Q | 26 |

## Change calculator

Difficulty: 🛠🛠🛠🛠  
Requires knowledge of objectives 1-7

Write a program that outputs the change to be given by a vending machine using the lowest number of coins. E.g. £3.67 can be dispensed as 1x£2 + 1x£1 + 1x50p + 1x10p + 1x5p + 1x2p.

## Parser challenge

Difficulty: 🛠🛠🛠🛠  
Requires knowledge of objectives 1-8

Write a program that allows the user to enter an addition in the format: 16+8. There may be any number of digits and additions in the input string. E.g. 16.4+108+9 is also a valid input. The parser should be a function that returns the answer.

## Blackjack challenge

Difficulty: 🛠🛠🛠🛠  
Requires knowledge of objectives 1-8

Create a program that plays a game of Blackjack between the player and computer. The program should:

* Deal two cards to the player and computer. One of the dealers cards is shown face up. All other cards are dealt face down.
* Cards 2-10 are face value. All picture cards are 10. Ace is 1 or 11, player may choose.
* Once the card have been dealt ask the user if they want to twist or stick.
* If the user twists then a new card is dealt and that is added to their total.
* If the player busts (goes over 21) the dealer wins.
* Once the player sticks, the dealer turns over his card and may twist or stick.
* The player or dealer cannot stick below 16.
* If the dealer busts then the player wins.
* If both players stick, the player closest to 21 wins.

## Lottery challenge

Difficulty: 🛠🛠🛠🛠  
Requires knowledge of objectives 1-8

In a lottery, players pick 6 numbers between 1 and 59. Six unique random balls are then output, together with a “bonus ball”. Prizes are awarded for matching 2, 3, 4, 5, 5 and the bonus ball or 6 numbers. Write a program to simulate the National Lottery, outputting from 1,000,000 draws how many times the player won each of the prizes. Use your program to prove the odds of winning the jackpot of 6 numbers is 1 : 45,057,474.

## Pass the Pigs challenge

Difficulty: 🛠🛠🛠🛠  
Requires knowledge of objectives 1-8

Pass the pigs is a game where a player tosses two plastic pigs and scores points according to how they land. If the two pigs land on the same side, that is known as a ‘sider’ and the player scores 1 point. If one pig lands on its feet, this is known as a ‘trotter’ and the player scores 5 points. Both pigs landing on their feet is a ‘double trotter’ and scores 20 points. After each throw the player can throw again to increase their score. However, if both pigs land on opposite sides that is a ‘pig out’ and the player’s score is reset to 0. The player attempts to score 100 points or more in as few throws as possible.

|  |
| --- |
| **Pseudocode** |
| 1.0 Set player’s score to be 0 |
| 2.0 Loop while the player’s score is less than 100 |
| 2.1 Wait for the user to press enter |
| 2.2 Output a blank line |
| 2.3 PigA is a random number between 1 and 3 |
| 2.4 PigB is a random number between 1 and 3 |
| 2.5 Check if PigA and PigB land on the same side |
| 2.5.1 Output ‘sider’ – 1 point |
| 2.5.2 Increase player’s score by 1 |
| 2.6 Check if PigA and PigB both land on the other side |
| 2.6.1 Output ‘sider’ – 1 point |
| 2.6.2 Increase player’s score by 1 |
| 2.7 Check if PigA and PigB land on the opposite sides |
| 2.7.1 Output ‘pig out’ – back to 0 points |
| 2.7.2 Set player’s score to 0 |
| 2.8 Check if PigA and PigB land on the opposite sides other way |
| 2.8.1 Output ‘pig out’ – back to 0 points |
| 2.8.2 Set player’s score to 0 |
| 2.9 Check if PigA but not PigB lands on its feet |
| 2.9.1 Output ‘trotter’ – 5 points |
| 2.9.2 Increase player’s score by 5 |
| 2.10 Check if PigB but not PigA lands on its feet |
| 2.10.1 Output ‘trotter’ – 5 points |
| 2.10.2 Increase player’s score by 5 |
| 2.11 Check if PigA and PigB both land on their feet |
| 2.11.1 Output ‘double trotter’ – 20 points |
| 2.11.2 Increase player’s score by 20 |
| 2.12 Output player’s score |

## Pass the Pigs challenge part 2

Include these additional point scores:

|  |  |  |
| --- | --- | --- |
| Snouter | Pig lands on its nose | 5 points |
| Double Snouter | Both pigs land on their nose | 20 points |
| Razorback | Pig lands on its back | 5 points |
| Double Razorback | Both pigs land on their back | 20 points |

The game is now unbalanced because it is easier to score points than to pig out or score a sider. Change the balance of the game so it remains a challenge. Generate a higher random number and use a greater or lesser range of numbers to represent each position.

## Pass the Pigs challenge part 3

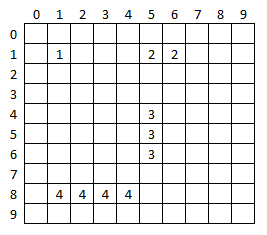
Create a two player version of the game where players can choose to bank after each throw, giving their opponent a chance to throw. Players can continue their go until they pig out or choose to bank. The first to 100 banked points wins.

Research the full range of scores available in the real game ‘pass the pigs’. Include these in your game with an appropriate balance of probability.

## Battleships challenge

Difficulty: 🛠🛠🛠🛠  
Requires knowledge of objectives 1-9

Battleships is a game for two players. Each player places four ships on a board but does not reveal their location to their opponent. Each ship occupies one or more adjacent squares either horizontally or vertically. Each player takes it in turn to pick a grid reference. The player scores a hit if the number matches a space occupied by a ship, or a miss if it does not. The player to sink all their opponents ships first wins.



* Create a one player game of battleships against a computer opponent.
* Keep score to tell the player how many hits and misses they have had.

## Battleships challenge part 2

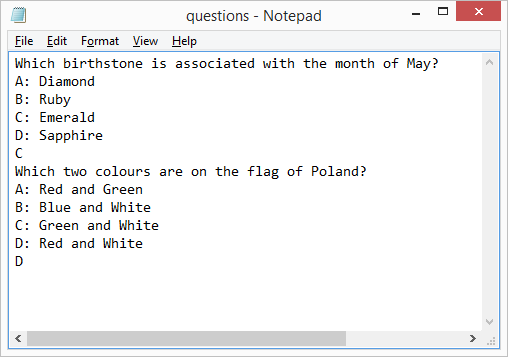
The computer chooses indexes to fire at more methodically. If one index is a hit then it should fire at the one before or the one after next until the ship is sunk.

## Quiz challenge

Difficulty: 🛠🛠🛠🛠

Q10 is a simple quiz game. Ten random questions are presented to the player one at a time, together with 4 possible answers. The player has to choose which answer: A, B, C or D is correct. The computer outputs the correct answer after the player has answered.

Create the following sample quiz file containing two questions in Notepad and save it in a folder for your new quiz program. Make sure you name it questions.txt



Note the file contains the question, the four possible answers followed by the correct answer.

Create the quiz program to use this file.

* The program outputs if the user got the question right or wrong.
* The program keeps score and outputs the score to the user.
* Modify the program to work for ten questions. Use a loop to show the questions one at a time, you don’t need to duplicate the code!
* The program shows the question number 1-10.
* The player is only allowed to answer the next question if they got the previous one correct. When an incorrect answer is given the quiz ends.
* The program picks a random set of questions stored in different files.
* The program prevents an illegal data entry. Only characters A-D are accepted.

## Till challenge

Difficulty: 🛠🛠🛠🛠

Using the product catalogue program, create a program to meet the following specification:

1. A till can be put into ‘admin’ or ‘trading mode’.
2. In the trading mode, the user can perform the following operations:
   1. Enter a product code. The product is found and the description and price is displayed.
   2. The price of the product is added to a sub-total.
   3. When a product code is not entered, the total is shown.
   4. The user enters how much money was given and the change is shown.
   5. A new set of transactions then begins.
3. In the admin mode, the user can perform the following operations:
   1. View the product catalogue
   2. Add a new product
   3. Edit a product
   4. Delete a product

## London Underground challenge

Difficulty: 🛠🛠🛠🛠  
Requires knowledge of objectives 1-9

The only station on the London Underground that can be formed without using any of the letters in the word mackerel is St John’s Wood. This is also true for the words piranha and sturgeon, although for different stations.

For a given list of stations, write a program that takes a word and determines if there is a single station that can be formed without using any of its letters.