

Computing Studies

Software Design
and Development

Programming Theory



&



Buckhaven High School



Version 1

Contents

Page 2	How to use this booklet
Page 3	Introduction
Page 4	High Level Languages
Page 6	Designing Computer Programs Flow Charts
Page 8	Pseudocode Structure Charts Agile
Page 9	Programming Constructs
Page 10	Programming Constructs and Pseudocode Assigning Values to Variables
Page 11	Arithmetic Operations
Page 12	Concatenating Strings
Page 13	Simple Selection Constructs
Page 14	Complex Selection Constructs
Page 15	Iteration and Repetition
Page 16	Pre-Determined Functions
Page 18	Identifying Programming Constructs in Pseudocode
Page 20	Testing
Page 21	Errors in Programs
Page 22	Readability of Code
Page 23	Translating Programs

How to use this booklet

This booklet has been written to cover the following content in National 4 and National 5 Computing.

	National 4 	National 5 
Computational Constructs	Exemplification and implementation of the following constructs: <ul style="list-style-type: none"> expressions to assign values to variables expressions to return values using arithmetic operations (+, -, *, /, ^) execution of lines of code in sequence demonstrating input - process - output use of selection constructs including simple conditional statements iteration and repetition using fixed and conditional loops 	Exemplification and implementation of the following constructs: <ul style="list-style-type: none"> expressions to assign values to variables expressions to return values using arithmetic operations (+, -, *, /, ^, mod) expressions to concatenate strings and arrays using the operator use of selection constructs including simple and complex conditional statements and logical operators iteration and repetition using fixed and conditional loops pre-determined functions (with parameters)
Data Types and Structures	string numeric (integer) variables graphical objects	string, character numeric (integer and real) boolean variables 1-D arrays
Testing and Documenting Solutions	<ul style="list-style-type: none"> normal, extreme and exceptional test data readability of code (internal commentary, meaningful variables names) 	<ul style="list-style-type: none"> normal, extreme and exceptional test data syntax, execution and logic errors readability of code (internal commentary, meaningful variables names)
Algorithm Specification		Exemplification and implementation of algorithms including <ul style="list-style-type: none"> input validation
Design Notations (also applies to ISDD)	<ul style="list-style-type: none"> graphical to illustrate selection and iteration other contemporary design notations 	<ul style="list-style-type: none"> pseudocode to exemplify programming constructs other contemporary design notations
Low Level Operations and Computer Architecture		Translation of high level program code to binary (machine code): interpreters and compilers

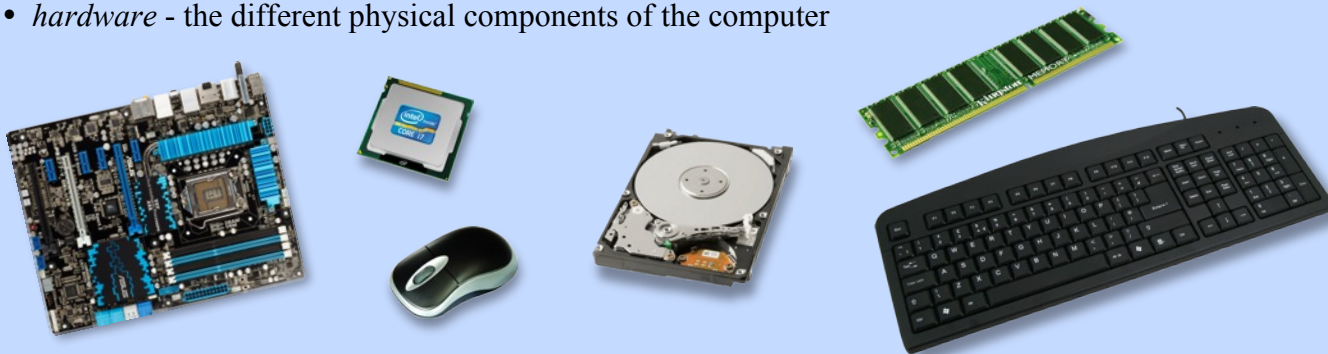
The booklet is colour coded as shown above.

For assessment purposes, pupils working at National 4 level should revise only the N4 content. Pupils attempting National 5 assessments, coursework or final exam should study only N5 content. (N5 pupils may wish to revise N4 content anyway to improve their overall knowledge of the subject.)

Introduction

To create a functioning, useful computer two things are required:

- *hardware* - the different physical components of the computer



- *software* - the programs that run on the computer

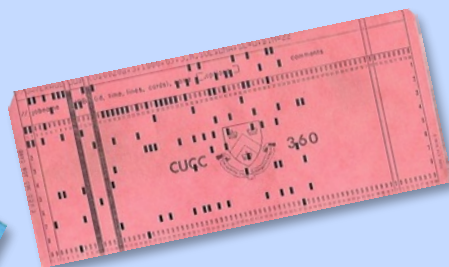
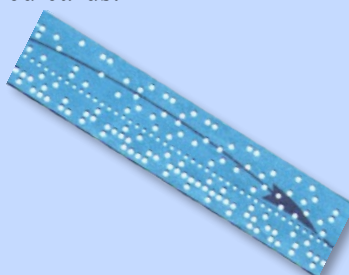


Programs are lists of instructions that tell the computer's hardware what to do. Without instructions (or programs) a computer is a useless collection of electronics.

Instructions found in computer programs are vast and varied but they essentially fall into one of the following categories:

- inputting data
for example - getting input from a keyboard or receiving a reading from a sensor
- outputting data
for example - sending an image to a monitor or switching on a motor
- processing data
for example - making a decision (is 10 larger than 6) or performing a calculation

In the early days of computing, programs would be written in binary. The binary 1s and 0s would be entered via switches, punched tape or punched cards.



High Level Languages

Writing lengthy binary programs is very time consuming and extremely difficult to get right. A single mistake in a 0 or 1 would cause the program to fail and is almost impossible to find.

```

0001 0001 0000 0011 0000 0101 0101 0110 0111
1000 0000 0000 0011 0000 0101 0101 0101 0101
0110 0111 1000 0001 0001 0000 0011 0100 0100
0101 0110 0111 0000 0000 0000 0000 0100 0101
0110 0111 1000 0000 0000 0011 0000 0101 0101
0101 0101 0110 0111 0000 0000 0000 0011 0100
0100 0101 0000 0111 0000 0000 0000 0011 0000
0000 0000 0000 0000 0001 0001 0001 0010 0011
0011 0100 0101 0110 0111 1000 0001 0001 0100
0011 0100 0101 0110 0111 0000 0000 0000 0011
0011 0011 0100 0101 0110 0111 0000 0000 0000
0011 0000 0101 0101 0101 0101 0110 0111 1000
0000 0000 0011 0011 0000 0101 0101 0101 0101
0110 0111 0000 0000 0000 0000 0000 0101 0101
0101 0101 0110 0111 1000 0000 0000 0000 0100
0000 0110 0111 0000

```

A Binary Program

Today's computer programs are written in English using an editing program. The English instructions are then translated into binary instructions that the computer can understand.

The example below shows a Python program written in English.

```

1 length = [0.0]*20
2 width = [0.0]*20
3
4 areaRoom = 0.0
5 totalArea = 0.0
6
7 noOfRooms = int(input("Please enter the number of rooms in the house"))
8
9 for loop in range(noOfRooms):
10     length[loop] = float(input("Please enter the length"))
11     width[loop] = float(input("Please enter the width"))
12
13 print("The total area is calculated as:")
14
15 for loop in range(noOfRooms):
16     print("Room",loop+1)
17     areaRoom = length[loop] * width[loop]
18     print(width[loop],"x",length[loop],"=",round(areaRoom,2),"metres squared")
19     totalArea = totalArea + areaRoom
20
21 print("Total Area =",round(totalArea,2),"metres squared")

```

A Python Program

Python is one of hundreds of programming languages that use English instructions. Programming languages that use English based instructions are called 'high level languages'.

Each language has its own set of rules describing exactly how each instruction should be written. This is called the 'syntax' of the programming language.

Task 1 - Different Syntax, Same Result

When users learn to program they often start with the instruction required to display, or print, the text “Hello World” on the screen.

If you search the world wide web for “print hello world in Python” you will find that the syntax for the instruction to do this is:

```
print(“Hello World”)
```

Your task is to research the correct syntax required to display “Hello World” in all the programming languages listed below.

Copy and complete the table below.

Name of programming language	Syntax to display “Hello World”
Python	print (“Hello World”)
Java	
C	
Visual Basic	
PHP	
Javascript	

E-mail your completed file to your teacher.

Novice high level language programmers have to learn two things at the same time.

- *language syntax* - how to write/format each type of instruction, where to put commas, brackets, spaces, indentation etc
- *problem solving* - how to put instructions together to make a program do what you want it to do

This can be a very challenging, frustrating process. A lot of practice is required to become a good computer programmer.



Designing Computer Programs

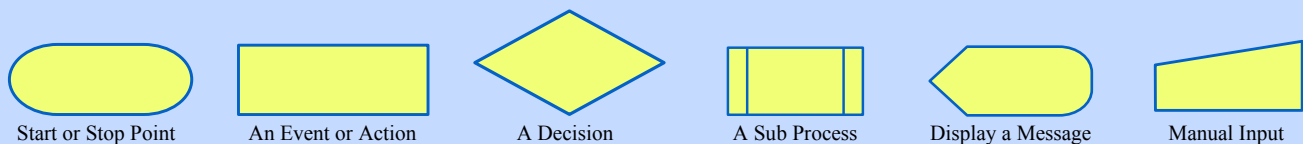
A good programmer develops the ability to take a problem apart and design a method of solving it. The solution must be developed in such a way that it can be programmed. This type of problem solving is called *computational thinking*.

With larger or more complex problems, programmers may use a variety of methods to design their program.

Flow Charts

A *flow chart* may be used to sketch out the order in which events take place. Flow charts are good at showing clearly when decisions are made in a problem as the flow chart will branch in two or more directions.

The symbols in a flow chart represent either information or events.

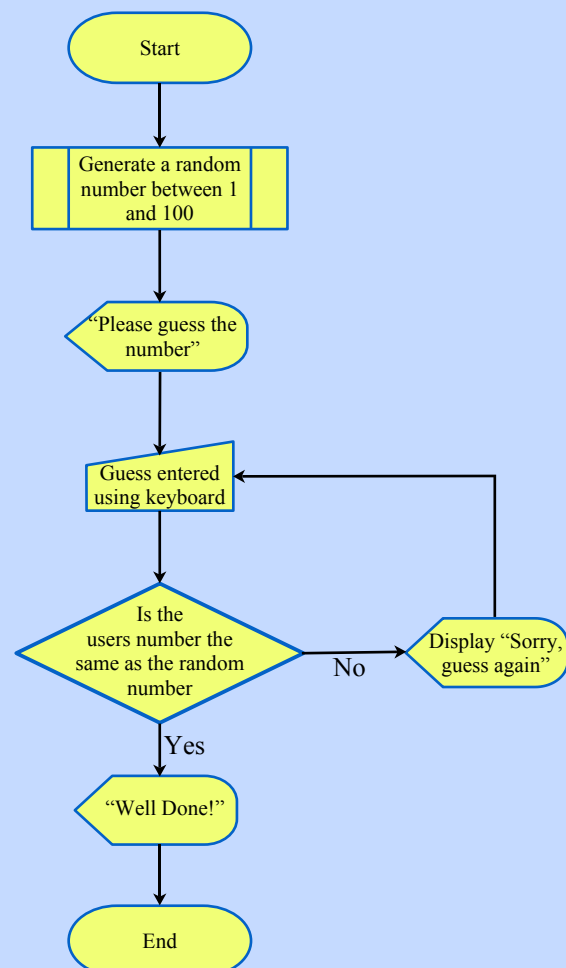


The symbols are connected by arrows, representing the flow from start to end. →

This flow chart to the right is a design for a game where the user has to guess a random number between 1 and 100.

The flow chart shows:

- a start
- a sub process where the computer generates the random number
- three messages displayed to the user of the program
- one manual input from the keyboard
- a decision - is the guess right or wrong
- an end

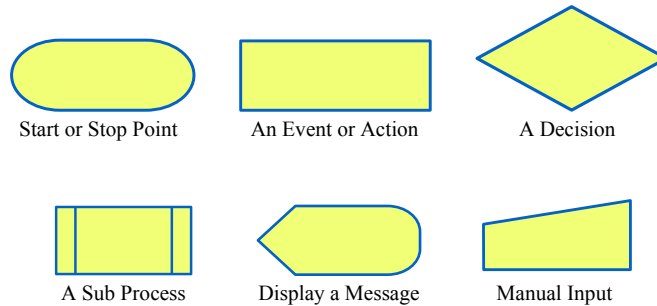


Task 2 - Flow Chart Problems

Initial designs for a program may often be a scribble on a piece of blank paper. By scribbling down their first ideas, a programmer can think through the problem and then create a more formal design.

On a sheet of blank paper, draw out a design for the three problems below. Use pencil as you may often change your mind.

Remember the basic flow chart symbols.



Problem 1

A program is needed to allow Nursery children to test their animal knowledge. The program will:

- show a random picture of an animal
- ask if the animal is awake during the nighttime or the daytime
- add 1 point on to their score if the pupil answers correctly
- when the pupil scores 10 points don't show any more pictures
- show a message congratulating the pupil for getting 10 correct answers.

Problem 2

A computer program asks its user to enter the age of each member of their family one at a time. The program will add each age onto a total. The program will stop asking for ages to be entered when the user enters 0. When all the ages have been entered the program will display the average age of the family members.

Problem 3

A program is required calculate a player's score in the card game Hearts. The player scores 1 point for each card in their hand that is a heart. There are 13 cards in a hand.



Pseudocode

Another common design method used in programming is *pseudocode*.

The term pseudocode comes from the word:

‘pseudo’ - pretending to be something else

‘code’ - as in program code

Pseudocode designs look very similar to program code but pseudocode is written in natural language (just as you would normally write in English) and has none of the strict syntax rules associated with a programming language.

An example of a pseudocode design is shown below.

```
Line 1  SET totalRunningTime TO 0
Line 2  <Get valid numberOfTracks from user>
Line 3  FOR counter FROM 1 TO numberOfTracks DO
Line 4    <Get title and length from user>
Line 5    SET totalRunningTime TO totalRunningTime + trackLength[counter]
Line 6  END FOR
Line 7  <display track titles and track lengths>
Line 8  SEND ["CD-R running time      " & totalRunningTime] TO DISPLAY
```

```
Line 2.1 REPEAT
Line 2.2   RECEIVE numberOfTracks FROM (INTEGER) KEYBOARD
Line 2.3   IF numberOfTracks < 1 OR numberOfTracks > 20 THEN
Line 2.4     SEND "Please enter number of tracks between 1 and 20" TO DISPLAY
Line 2.5   END IF
Line 2.6 UNTIL numberOfTracks >= 1 AND numberOfTracks <= 20
```

```
Line 4.1 RECEIVE trackTitle[counter] FROM (STRING) KEYBOARD
Line 4.2 RECEIVE trackLength[counter] FROM (REAL) KEYBOARD
```

```
Line 7.1 FOR counter FROM 1 TO numberOfTracks DO
Line 7.2   SEND [trackTitle[counter] & trackLength[counter]] TO DISPLAY
Line 7.3 END FOR
```

Algorithm

Refinements

An *algorithm* is an outline of how a problem is solved. This is shown in lines 1 to 8.

Three of the lines (2, 4 and 7) have been expanded to show more detail about how these sub-problems would be solved. These are called *refinements*.

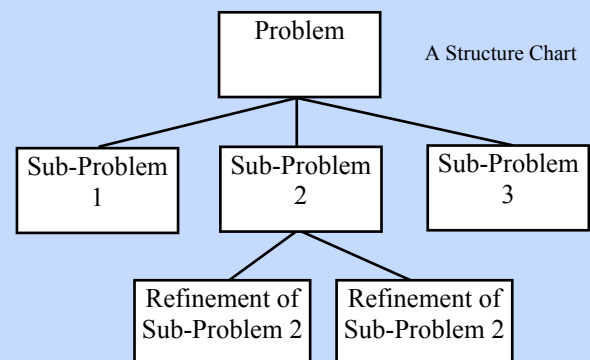
The process of starting with an algorithm that is expanded is called ‘*top-down design*’.

Other Design Methodologies

Structure charts are another method of top down design. They use blocks to show how a larger problem is broken up into sub-problems.

Agile software development is a design method that avoids paperwork. Teams working on a agile project work in a close groups (called bullpens) relying on regular face-to-face meetings. Each team contains all the necessary skills to design and develop a project.

The aim of agile development is produce a working piece of software very quickly (about 3 to 4 weeks) in blocks of time called iterations. Each iteration becomes a miniature project on its own. After each iteration the team will meet to discuss what should be produced next.



Programming Constructs

A program is constructed from building blocks called constructs. When designing a computer program, the designer will consider these building blocks as they construct a plan of attack.



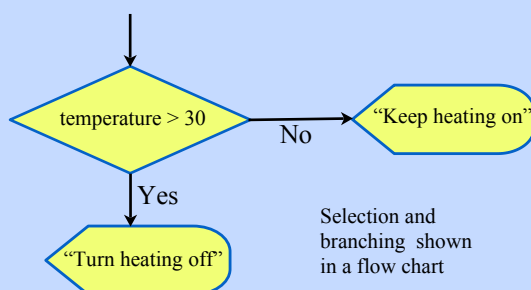
Common constructs are:

- Assigning values to variables**
 Variables are used in programs to store values (text and numbers). To 'assign' a value to a variable simply means to store a value in that variable.
- Arithmetic operations**
 Programs input, process and output data. Often the processing takes the form of a calculation or 'arithmetic operation'. For example, calculating an average of several values.

- Selection constructs**
 These constructs are used to make decisions by comparing values. These comparisons are also known as *conditions*. Some examples of conditions are shown below:

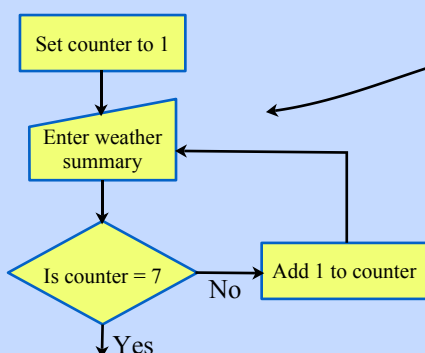
$11 > 2$
 number ≤ 27
 word = "Edinburgh"

The program may carry out different processes if the conditions are found to be true or false. This is called branching as the program 'branches' in different directions.



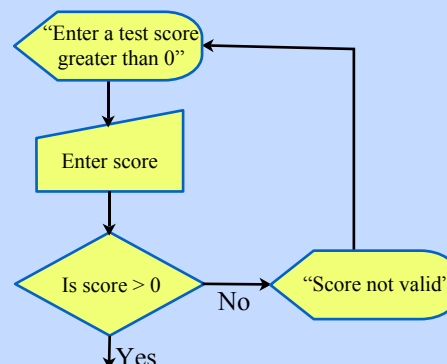
Selection and branching shown in a flow chart

- Repetition**
 Programs often repeat instructions. There are two types of repetition in programs:



Unconditional repetition is when a program is written to repeat a set number of times. For example if a program needed to stored a value for each day of a week it would ask 7 times for a value to be entered.

Conditional repetition is found when a program keeps repeating until a condition is true. For example, if a program asks a user to enter a test score greater than 0, conditional repetition could be used to keep asking for the score to be entered until the user enters a valid score.



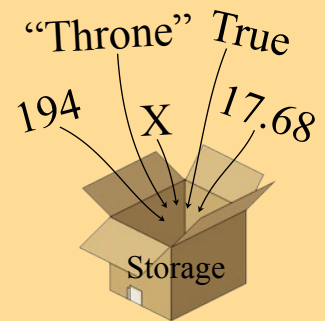
Programming Constructs and Pseudocode

When designing a program, the programming constructs used to solve the problem will be written out in the pseudocode. Examples of each construct are shown over the next few pages.

• Assigning Values to Variables

There are several types of data stored and processed by computer programs.

<i>Integers</i>	whole numbers
<i>Reals</i>	numbers with decimal places
<i>Characters</i>	a single letter, punctuation symbol or number
<i>Strings</i>	a group of characters, usually a word or sentence
<i>Boolean</i>	a true or false value



Arrays To store multiple integers or multiple strings we would use an array structure. For example, to store an list of 100 names we would use 'an array of strings'.

The pseudocode for an assignment should show:

- *Where* the value is being stored.
This will be the name of the variable or array the value will be stored in.
- *Type* of value being stored.
Is it an integer, real, character, string or boolean value.
- *How* the value is assigned.
Is the data entered by a user (probably using a keyboard) or is it assigned by the program itself.

Examples

SET number TO 973

Where - The value is being stored in a variable called 'number'.

Type - An integer (973) is being stored.

How - The program is storing the value.

SET name TO "Greg"

Where - The variable is called 'name'.

Type - String (Greg).

How - The program is storing the value.

SET score[3] TO 12.6

Where - The value is being stored in an array called 'score'. The [] tell us that it is an array structure.

Type - Real (12.6).

How - The program is storing the value.

RECEIVE stockItem FROM (STRING) KEYBOARD

Where - The variable is called 'stockItem'

Type - String.

How - The user is entering the value using a keyboard.

RECEIVE temperatureReading FROM (REAL) SENSOR

Where - The variable is called 'temperatureReading'

Type - Real.

How - The value is created by a sensor and passed to the computer.

Task 3 - Writing Assignment Statements in Pseudocode

Using pseudocode, write a design for the following small problems. Remember to consider *where*, *type* and *how* in your pseudocode statements.

1. A program stores that a car's fuel consumption is 56mpg.
2. A program stores that a user purchased 124 collectable cards last week.
A program inputs the number of cards purchased using the keyboard.
3. A program uses keyboard entry to store the names of 50 choir members in an array. Show how the 5th name would be stored in the array.



• Arithmetic Operations

As stated earlier, all programs process values and this often takes the form of a calculation. Calculations may be part of an output statement (sending an answer directly to a monitor) or an assignment, as the answers to the calculations are often stored in variables.

There are a few types of calculation commonly seen in programs.

Simple 12+3, 4*4, 5/8, 45-34

These usually involve a couple of values.

Complex (75+3)/(34*56)

These may involve a few separate calculations. Note that the normal rules you learned in maths apply in these example (*, / before +, - and calculate the parts inside brackets first)

Variables totalScore/12, testOne+testTwo

Calculations involving other variables.

Functions mod(35/4), int(75.3), round(99.52583,3), 25^2

Functions perform extra tasks in calculations like rounding or squaring numbers.

The pseudocode should show:

- *Where* the answer to the calculation being stored or where it is being sent.
This will be the name of the variable or a device like a monitor or printer.
- *Calculation* being carried out.
The details of the calculation being performed using values or other variables.

Examples

SET price TO 45+34/6

Where - The answer is being stored in a variable called 'price'.

Calculation - 45+34/6.

SET total TO num1 + num2

Where - The answer is stored in 'total'.

Calculation - num1 + num2. This calculation adds together the values stored in two other variables.

SEND 2^20 TO DISPLAY

Where - The answer is displayed on the computers monitor.

Calculation - 2^20 (or 2 to the power of 20)

Task 4 - Writing Calculation Statements in Pseudocode

Using pseudocode, write a design for the following small problems. Remember to consider *where* and *calculation* in your pseudocode statements.

1. A line of program code is required to find and store the answer to the following calculation: 12 multiplied by 45 plus 6
2. A program is required to calculate and store the cost of 10 bikes costing £125 each.
3. A program asks the user to enter the number of 1p coins they have collected in a jar. The program should then display the amount they have saved in pounds.
(Note - this requires 2 lines of pseudocode)
4. A program asks its user to type in their age and then also type the number of months since their birthday. A calculation is then performed to work out how many months they have been alive. The program should store the answer.
(Note - this requires 3 lines of pseudocode)
5. A program asks a user to enter the amount they have spent in one week and the amount of money they earned in the same week. The amount they saved will be sent to a printer.



- **Concatenating Strings**

Concatenation is when two or more strings or variables are joined together to make one string.

Concatenations may involve:

Strings Only

“Toy” & “Story”

Concatenated this would be “ToyStory” without a space between the two words.

Variables Only

forename & surname

The variables used would have to be string variables

Variables and Strings

“Age = “ & age

This is often used to create a message using text and stored variables.

The pseudocode should clearly show the difference between the name of a variable and a piece of text by using “”:

score is a variable

“score” is text

Examples

SET errorMessage TO [“Device failed due to ” & faultNumber]

one string and one variable

SEND [firstName & surname] TO DISPLAY

two variables

SET password TO [“agd” & “12” & “k jy”]

three strings

Task 5 - Concatenation Statements in Pseudocode

As before you will be asked to write pseudocode designs for the following problems. The problems will require assignment, arithmetic and concatenation statements in the one design.

1. A program asks its user to enter and store the number of times they brushed their teeth this week and the number of minutes they brush their teeth for. The two numbers should then be multiplied together to calculate the total number of brushing minutes. The answer should be displayed along with a suitable message.
2. A program is required to store a username for a website. The program will ask the user to enter their favourite word along with the day and month they were born. The username will be generated by joining the word to the day multiplied by the month. (For example - 'trepidation, 20, 5' would give a username of 'trepidation100')

• Simple Selection Constructs

Selection is required when a program must make a decision whether or not to execute a line (or block) of code. A simple selection (often called a '*condition*') involves the use of *operators* to compare two values or variables.

A list of operators are shown below:

=	Equal to
>	Greater than
<	Less than
>=	Greater than or Equal to
<=	Less than or Equal to
<>	Not equal to

Selection constructs may be found in *IF statements* and conditional loops. The examples below will only look at IF pseudocode statements as conditional loops are covered later.

Examples

`IF temperatureNow >= 100 THEN SEND "Water is in a gaseous form at this temp" TO DISPLAY`

The above example compares the value stored in the variable 'temperatureNow' to 100. If the value stored in the variable is greater or equal to 100 (if the conditions are 'true') the message is displayed.

```
IF temperatureNow >= 100 THEN
    SEND "Water is in a gaseous form at this temp" TO DISPLAY
    SEND "This is called steam" TO DISPLAY
END IF
```

If a selection statement is true, more than one line of code may be executed as shown above.

```
IF temperatureNow >= 100 THEN
    SEND "Water is in a gaseous form at this temp" TO DISPLAY
ELSE
    SEND "Water is a liquid or a solid at this temp" TO DISPLAY
END IF
```

A simple IF statement executes an instruction if the conditions are true, the above example uses an ELSE statement to execute another instruction if the conditions are not true (false).

• Complex Selection Constructs

A complex selection construct involves two or more conditions. The conditions are joined together using the logical operators shown below:

<i>AND</i>	Both conditions must be true
<i>OR</i>	Either condition may be true
<i>NOT</i>	The condition must be false

Again complex selection constructs may be found in *IF statements* and conditional loops. The examples below will only look at *IF pseudocode statements*.

Examples

```
IF temperatureNow > 0 AND temperatureNow < 100 THEN
    SEND "Water is in a liquid at this temp" TO DISPLAY
END IF
```

The above has two conditions. Both conditions must be true before the message is displayed.

```
IF temperatureNow <= 0 OR temperatureNow >= 100 THEN SEND "Water is not a liquid at this temp" TO DISPLAY
```

If either selection statement is true the message is displayed.

```
IF NOT(temperatureNow > 0) THEN
    SEND "Water is solid at this temp" TO DISPLAY
END IF
```

NOT conditions are harder to follow. The program would check to see if the condition inside the brackets is true. If the condition is true (the temperature is greater than 0) the message isn't displayed as the NOT operator states that it must now be false to be displayed.



Task 6 - Selection Constructs in Pseudocode

All of these problems will involve writing pseudocode designs with simple or complex selection constructs.

1. A program asks its user if they wish to enter more data. The user should enter Y or N. Design a single line of pseudocode that will display a message stating "Are you sure?" if the user enters N.
2. A program uses two variables to store two players scores (player1score and player2score). Write a simple selection statement that will display a suitable message if player 1 has a greater score than player 2.
3. A program decides if a car salesperson has earned a bonus. A message saying "bonus earned!" should be displayed if the employee has sold more than 50 cars. A bonus of £500 should also be stored using a suitable variable name.
4. Show how could the pseudocode in problem 3 can be expanded to display a message "no bonus" and store a bonus of £0 if the salesperson does not sell 50 cars.
5. A program asks a user if they wish to enter more data. The user should enter Y or N. Design a line that will display an error message if the user does not enter Y or N.
6. A quiz asks a user what the chemical symbol for Oxygen is. Design a line that displays "Sorry, you are incorrect" if the user enters anything other than O

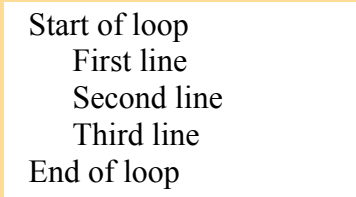


• Iteration and Repetition

It is common for programs to repeat lines of code. There are two types of repetition in programs that can be represented using pseudocode:

- Unconditional (or Fixed) Loop* This is when the number of times the program code will be repeated is known.
- Conditional Loop* This is when the code is repeated until a selection construct (a condition) is found to be true.

The pseudocode should clearly show the start and end of the loop using *indentation*, where the middle lines of the pseudocode (the ones being repeated) are moved in a bit from the left.



Unconditional Loop Examples

```

FOR loop FROM 1 to 10 DO
  RECEIVE nextInput FROM (REAL) KEYBOARD
  SET totalCost TO totalCost + nextInput
END FOR
    
```



The above loop counts from 1 to 10 (using a variable called 'loop' to store the place in the count) repeating the two lines in between 10 times. This is therefore an example of an unconditional loop.

```

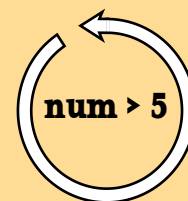
REPEAT 10 TIMES
  RECEIVE nextValue FROM (REAL) KEYBOARD
END REPEAT
    
```

This example of an unconditional loop also repeats the code 10 times but does not have a variable to count the place in the loop.

Conditional Loop Examples

```

REPEAT
  RECEIVE pressureLevel FROM (REAL) SENSOR
UNTIL pressureLevel >= 0 AND pressureLevel <= 200
    
```



The loop has conditions at the end of the loop (post-condition). The code in the loop will always be executed once before the conditions determine whether or not the code will then be repeated.

```

WHILE pressureLevel > 70 DO
  RECEIVE pressureLevel FROM (REAL) SENSOR
END WHILE
    
```

This conditional loop has its conditions at the beginning of the loop (pre-condition). This loop will keep repeating while pressureLevel is greater than 70. Note that if pressureLevel is already ≤ 70 the loop will never be executed and the program will move on to the next line.

Task 7 - Writing Repetition Statements in Pseudocode

Using pseudocode, write a design for the following line or lines of code.

1. A program asks its user to enter the number of hours of television they watch each day of the week. Each value they enter is added on to a total. Once all 7 values are entered the final total is displayed along with a suitable message.
2. A computer program is used to analyse the speed of a cyclist during a training exercise. The program uses a sensor to continually input the speed until the cyclist reaches 25mph. When the cyclist is travelling at 25mph or more a light is switched on to indicate this to the cyclist.
3. If you used a pre-condition loop for question 2 then rewrite your pseudocode using a post-condition loop.
Or, if you used a post-condition loop for question 2 then rewrite your pseudocode using a pre-condition loop.
4. A simple two player game is to be written as a computer program. Both players enter a number between 1 and 10. If the numbers are different 1 is added to a counter. The program will continue to ask for two numbers to be entered by the players until both players enter the same number. At the end of the game the counter is used to display a message stating the number of attempts the players had before they entered the same value as each other.



• Pre-Determined Functions

Pre-determined functions are built in features of programming languages that perform tasks or calculations. There are many common functions that can be found in most programming languages. These common functions are often used in program design methods, like pseudocode.

Common pre-determined functions:

<i>Modulus</i>	This mathematical function calculates the remainder when one number is divided by another.
<i>Length</i>	This function calculates the number of characters in a string and returns a integer.
<i>Integer</i>	This function removes the decimal places from a real number leaving just the integer (23.85 becomes 23)
<i>Lower Case</i>	Converts a string to all lower case letters.
<i>Upper Case</i>	Converts a string to all upper case letters.
<i>Ord</i>	Converts a single characters to its ASCII code number.
<i>Chr</i>	Converts an ASCII code value to its equivalent character.
<i>Round</i>	This will round a real number to a given number of decimal places.
<i>Random Number</i>	This function will generate a random number between given limits.

The pseudocode should clearly show the function being used followed by the value or variable the function is being applied to in brackets. For example `length(firstname)`.

Examples

SET lengthOfWord TO length("computing")

Calculates that there are 8 characters in "computing" and stores the result in the variable 'lengthOfWord'

SET numberToGuess TO random(1,100)

Generates a random number between 1 and 100 and stores the result in the variable called 'numberToGuess'.

SEND ["The average is:" & round(average,2)] TO DISPLAY

This examples rounds the value stored in 'average' to 2 decimal places, concatenates it with a message and displays the result.

```
RECEIVE selectedOption FROM (CHARACTER) KEYBOARD
WHILE ord(selectedOption) < 65 OR ord(selectedOption) > 70 DO
    RECEIVE selectedOption FROM (Character) KEYBOARD
END WHILE
```

This example shows how the ord function could be used to ensure that the user of a program enters a single character between A(65) and F(70).

Task 8 - Pre-Determined Functions Constructs in Pseudocode

All of these problems will involve designing several lines of pseudocode using all the constructs covered in the previous pages. Each problem will require at least one pre-determined function.

1. A program asks its user to select a new account name. The user is informed that the name should be at least 10 characters long. If the user enters a name of less than 10 characters they should be asked to enter another account name. This process should be repeated until an acceptable name is entered.
2. A Program is required to generate a 4 digit PIN number for new bank customers. The program should select 4 random numbers between 0 and 9. Each number should be joined to the previous number creating a 4 character string (2471). The PIN number should then be sent to the printer in order that it can be posted to the customer.
3. A program asks its user to enter the length and breadth from the plans of a garden deck in metres. Assuming the planks of wood for the decking are 2m long and 0.2m wide, design a program that will calculate the number of planks required to build the deck. An example output for the program is shown below:

Deck Area = 20.56 metres square
 Plank Area = 0.4 metres square
 Number of whole planks needed = 51
 Part of plank left over = 0.4



Identifying Programming Constructs in Pseudocode

Understanding a program design involves first identifying the purpose of each small part of the design. Like a jigsaw puzzle, once you see how the pieces fit together, you can build the larger picture of the purpose of the whole design.

Now that you have practiced writing pseudocode designs you should find it fairly easy to identify the different constructs in a larger design.

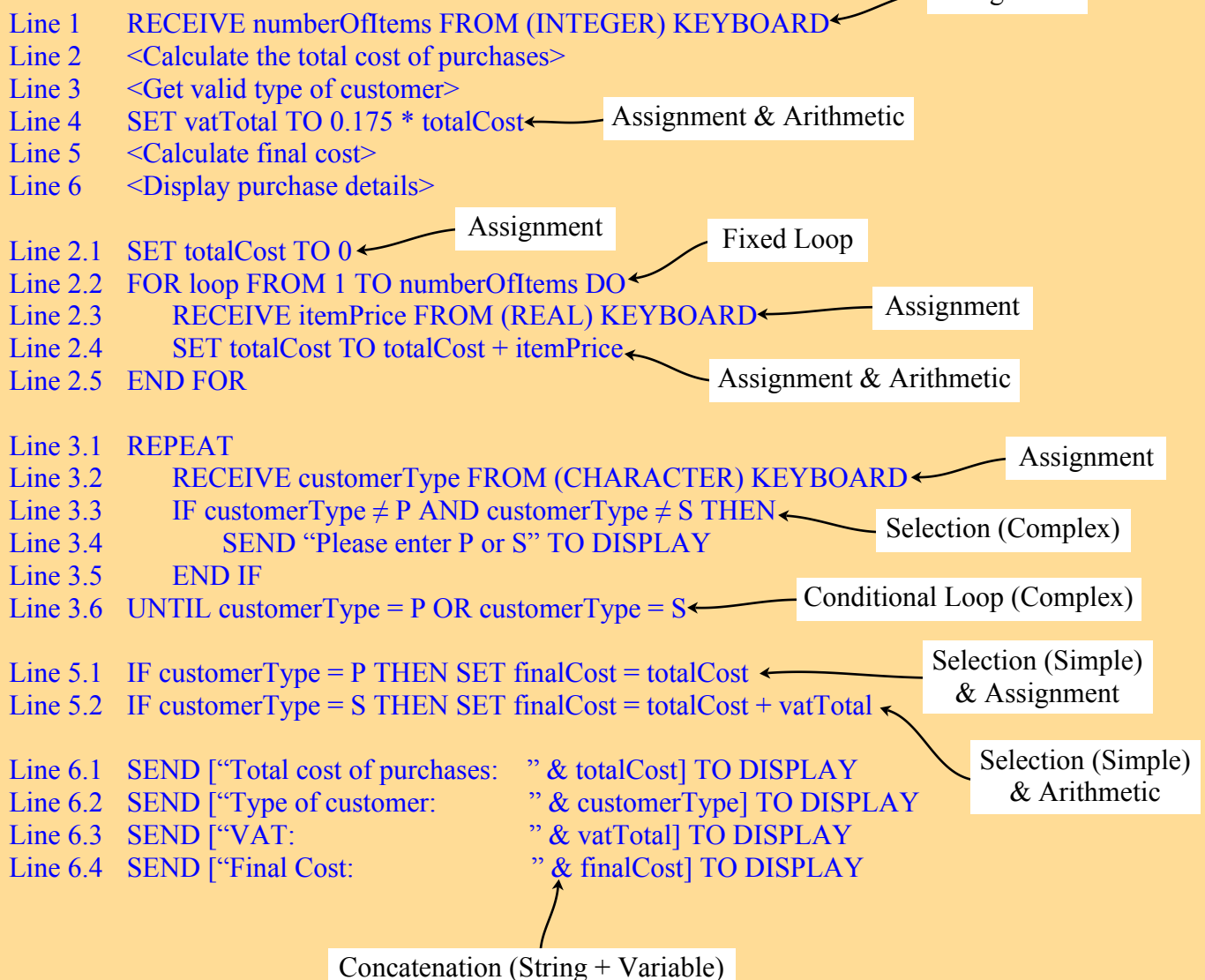


Worked Example

Problem - The manager of a school cafeteria wants to use a computer system to calculate how much each customer has to pay. Members of staff have to pay VAT on their purchases but pupils do not. If the customer is a member of staff then the program will calculate the VAT and add it to the total cost.

VAT is calculated using the formula: $VAT = 0.175 \times \text{total cost}$

Pseudocode



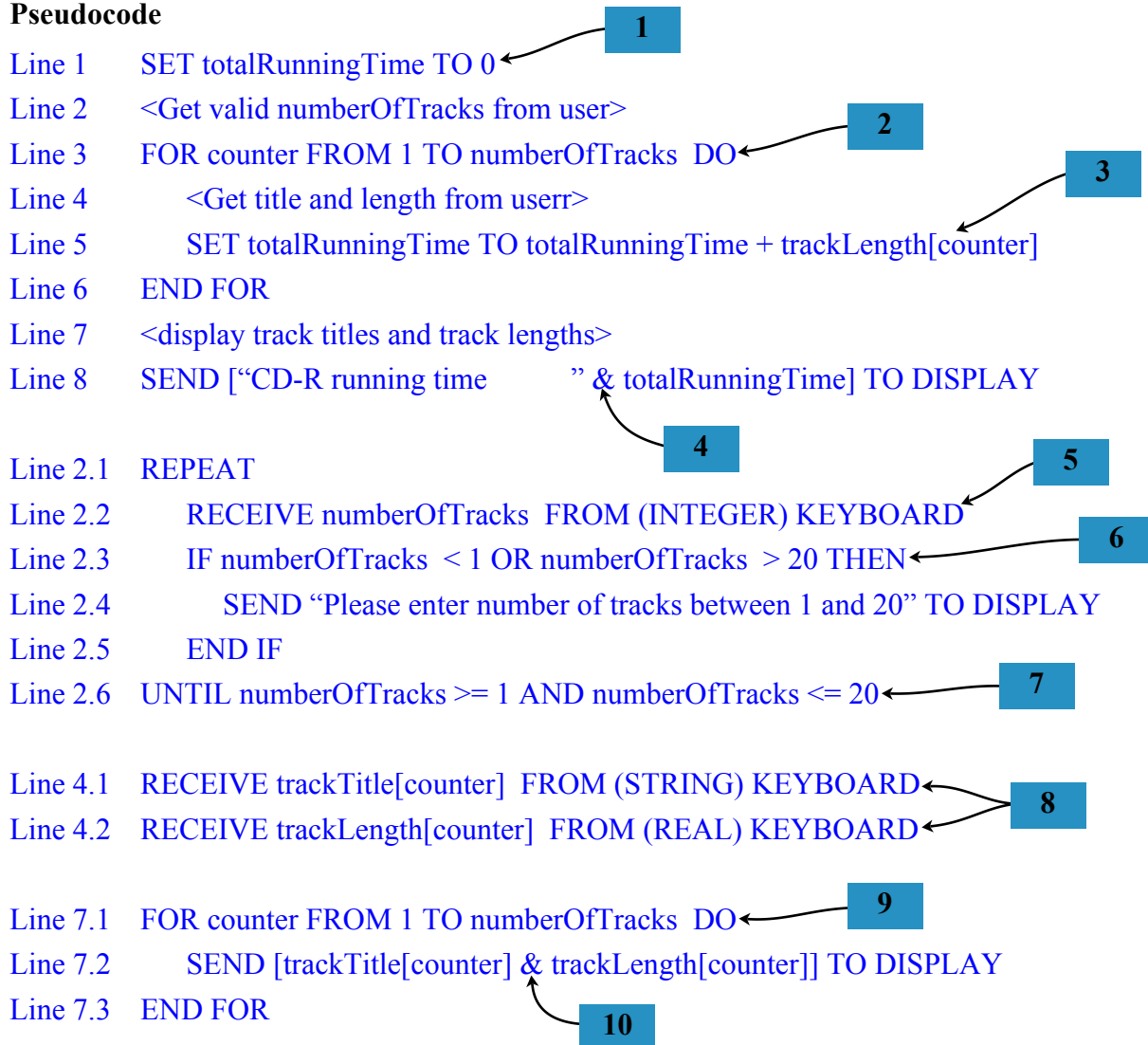
Task 9 - Identifying Programming Constructs

The pseudocode design below shows a main algorithm (lines 1 to 8) and refinements of lines 2,4 & 7.

Part 1

Your task is to identify the programming constructs labelled 1 to 10 next to the pseudocode.

Pseudocode



Part 2

Can you now piece together the jigsaw and describe the purpose of the whole design.

Email your answers for parts 1 and 2 to your teacher.

Testing

When a program is complete it should be tested thoroughly to prove that it can cope with a variety of different inputs without crashing.



Worked Example

A program is written to record rainfall (in millimetres) over the course of a month. The program expects each daily measurement to be entered as a number between 0 and 200. An error message is displayed if the user tries to enter a number outside these limits.

To test this program fully, the user should enter three types of input.

Normal Input the program would normally expect.
In this example this is any number from 0 to 200.

Extreme Input that is on the limits of what the program expects.
In this example the limits are 0, 200.

Exceptional Input that the program would not normally expect.
In this example this is any number less than 0 or greater than 200 (-5, 348 etc).
Note that exceptional data could also include the wrong type of data being entered. 'Banana' would also be an example of exceptional test data as the program is expecting a number to be entered.



When testing a program it is common to produce a test plan, mapping out how the program will be tested.

	Test Data	Expected Result (what you predict will happen when the data is entered)	Actual Result (when the data is entered into the program)
Normal	3 89 45.7	Input accepted Input accepted Input accepted	Input accepted Input accepted Input accepted
Extreme	0 200	Input accepted Input accepted	Input accepted Input accepted
Exceptional	-5 348 Banana	Input rejected Input rejected Input rejected	Input rejected Input rejected Input rejected

Task 10 – Creating Test Plans

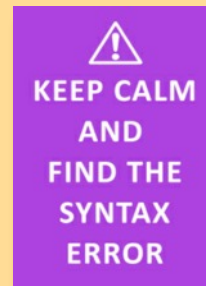
Using the above table as an example of the layout required, create a test plan (including normal, extreme and exceptional data) for each of the programs described below.

1. A program calculates the average distance travelled in a week by a cyclist. Each day is entered as a single value. The program assumes that the cyclist will travel no more than 100 miles in a single day.
2. A program is written to store the result of a tennis match. An example of a match result (showing the maximum and minimum values that can entered) is 6-4, 6-7, 6-0.
3. A password program allows a user to created a password that will later be used to access a database of club member's personal details. The password entered by the user should be a single word, 6 to 8 letters long.

Errors in Programs

Two types of errors occur in computer programs:

- errors that prevent the program being executed (syntax)
- errors that occur while the program is running (execution, logic)



Syntax Errors

Syntax errors occur during the creation of a computer program when an instruction is incorrectly typed or formatted. A syntax error will prevent the program from running as the editor can not convert into binary an instruction it can't understand.

<i>Misspelt Keywords</i>	Pront (“This will not work”)	This should have said ‘Print’ rather than ‘Pront’.
<i>Formatting Errors</i>	Print (“Missing something)	There should be another “ at the end of the message.
<i>Wrong Language</i>	writeln (“Another language”)	Programmers who work in several different languages at once may type a line that would work well in another program but appears as a syntax error in the language they’re currently working in.



Execution Errors

An execution error (often called a ‘runtime’ error) occurs while the program is running.

<i>Unexpected Input</i>	An example of unexpected input could be if the program is expecting the user to enter an Integer but they enter a string instead causing the computer program to crash.	
<i>Never Ending Loops</i>	If the conditions for a conditional loop are written in a way that they can never be true the program will enter the loop but have no way of leaving it. This will not crash the program but as the program can not go any further the user will have to quit from it.	
<i>Division by 0</i>	It is mathematically impossible to divide a number by zero as there is no answer. If this happens in a program it will crash.	

Logic Errors

Logic error are caused by poor design or poorly written code.

<i>Arithmetic Logic Errors</i>	For example, imagine a program that calculates the value of a bag of coins. The value 127, 1p coins would be calculated as: $\text{valueOf1pCoins} = \text{numberOf1pCoins} * 1$ If the programmer enters this calculation as $\text{valueOf1pCoins} = \text{numberOf1pCoins} + 1$ The program would run without any problems it just wouldn't give the correct answer.
<i>Selection Logic Errors</i>	Mistakes in conditional statements (num<10 instead of num>10) may cause a program to execute the wrong instructions.
<i>Sequential Logic Errors</i>	These types of errors are caused by lines of code being executed in the wrong order. For example, a calculation is carried out before the data is input.

Readability of Code

While writing code, programmers will use techniques to ensure that their code is easy to read.

This is important for the following reasons:

1. While editing the program, it is easier to find mistakes in readable code.
2. While editing the program it is easier to understand the purpose of each line or section of code.
3. When the program is finished it is easier for another programmer (or the same programmer) to return at a later date and add more to the program.



Program code can be made more readable in the following ways.

- *Comment lines* throughout the program explain the purpose of each line or section.
- *Meaningful variable names* make it clear what data the variable is storing.
- *White space* (blank lines) separate out sections of the code that perform different functions.
- *Formatting* the code makes keywords stand out. Many modern editors, like the one shown, do this automatically.

```

1 # Program 43
2 # Horse Hands
3
4 # The program stores the details of 6 horses
5 names = ["Black Beauty", "Red Run", "Bess", "Derek", "Phillis", "Mental"]
6 ages = [12, 18, 8, 5, 10, 13]
7 height = [13, 14, 12, 15, 16, 18]
8
9 maxAge = 0
10 maxHeight = 0
11
12 # The maximum age and height are input by the user
13 maxAge = int(input("Please enter the maximum age of the horse"))
14 maxHeight = int(input("Please enter the maximum height of the horse"))
15
16 # This loop displays only selected horses
17 for loop in range(8):
18     if ages[loop] <= maxAge and height[loop] <= maxHeight:
19         print (names[loop]+",", ages[loop], "years ,"+height[loop],

```

The above code, without the additional readability would look like this.

You can see how difficult it is to understand the purpose of the code if it was poorly written.

```

1 n = ["Black Beauty", "Red Run", "Bess", "Derek", "Phillis", "Mental"]
2 a = [12, 18, 8, 5, 10, 13]
3 h = [13, 14, 12, 15, 16, 18]
4 mA = 0
5 mH = 0
6 mA = int(input("Please enter the maximum age of the horse"))
7 mH = int(input("Please enter the maximum height of the horse"))
8 for l in range(8):
9     if a[l] <= mA and h[loop] <= mH:
10        print (n[l]+",", a[l], "years ,"+h[l], "hands")

```

Translating Programs

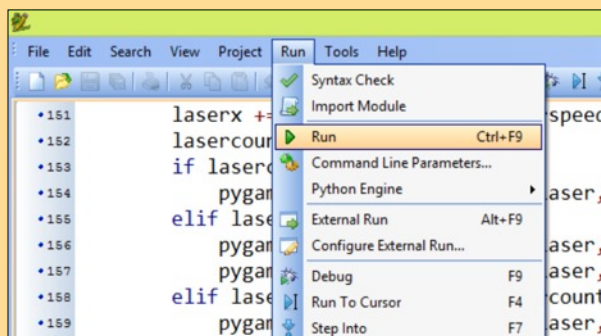
It is important to remember that, in their English form, computer programs written in a High Level Language will not run on any computer system. For a HLL program to be run the English program code must be translated into binary instructions before the code can be executed.

Program code in binary form is known as 'machine code'.

Translation is required at two stages during the development of a computer program.

Interpreters

During the creation of the program, the programmer will run the code many times as they write the code, to check that each new section of code works correctly. This will be done without leaving the program editing environment using an interpreter.



When the programmer selects run from a menu in the editor, the interpreter begins translating one instruction at a time, executing each line after it has been converted to a machine code instruction.

The editor will open an output window to allow input to be entered and output to be viewed.

When the run is complete no machine code remains as the translated instructions are not kept.

Compilers

Once a program is complete and fully tested it is translated using a compiler. A compiler translates the entire program creating a new executable file (.exe) that will run on it's own without the need for the editing environment.

HLL Code

```

1 # Basic Pygame Structure
2
3 import pygame
4 import random
5 import sys
6
7 black = ( 0, 0, 0)           # Define some colors using
8 white = ( 255, 255, 255)
9 laser = ( 110, 238, 243)
10
11 pygame.init()
12 size = [700,500]           # Set the width and height
13 screen = pygame.display.set_mode(size)
14 pygame.display.set_caption("Defender")   # Name your window
15 done = False              # Loop until the user click
16 clock = pygame.time.Clock() # Used to manage how fast
17
18 score = 0
19
20 background_image = pygame.image.load("SpaceBackground.jpg").convert()
21 player_image = pygame.image.load("ShipRight.png").convert()
22 player_image.set_colorkey(white)
    
```

Machine Code

```

0010101001001010101010100010
1010100010100101101100101100
10101011010101010101010101010
1010101010101011101010011010
110110001010100100101010101
010010101010010100101101100
10110010101011010101010101010
1010101010101010101011101010
011010110110001010100100101
01010100101010100010100101
    
```



Space Shooter.exe

Compiled, machine code programs will execute faster than interpreted English instructions as the computer does not have to translate and execute the instructions at the same time.

Task 11 - Interpreter vs Compiler

Answer the questions below and e-mail the answers to your teacher.

1. What advantage is there, to a programmer, of being able to run a program in the code editing environment.
2. Explain the statement "some binary is machine code but all machine code is binary".
3. What would you notice if you executed a fixed loop with 100,000 iterations in an interpreter and then compiled the loop and executed it again.