Tshwane University of Technology

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Principles of Programming A (Extended)

PPAF05D

Introduction to Programming 115R (Extended)

TROF05D

Unit 2

Basic programs in Scratch

You OWN your learning – therefore you need to reflect on your learning process. At the end of each lesson, you need to have a discussion with yourself and/or fellow students and/or lecturer to reflect on the lesson of the day. See the paragraph called ‘Reflection’ at the end of the chapter for examples of questions you can include in your discussions.

Unit 2: Basic programs in Scratch

Assessment criteria

The content covered in this unit relates to the following outcomes and assessment criteria:

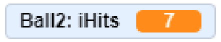
UNIT 2: Basic programming principles
Outcome: The student must be able to apply algorithmic problem solving to basic mathematical and programming problems.
Assessment Criteria
• Input and output can be identified for a given programming problem. • The variables needed as part of an algorithm can be identified by completing an input-processing-output (IPO) table. • Valid variable names can be provided following the best practices standards of the OOP language used. • Constants to be used in a program can be identified by writing it in the correct section of an algorithm. • Valid data types for variables can be suggested by indicating the data type in an algorithm. • The rules of the order of processing can be applied in writing pseudo code for a mathematical calculation. • A solution can be planned using an input-output-processing (IPO) table. • An algorithm to solve a basic mathematical problem requiring sequential steps can be designed, presented in pseudo code and implemented in a programming language providing correct output. • Comments can be added in a program to clarify the flow of the program..

Introduction to variables

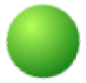
In algebra in Mathematics you have learned about using *variables*. Usually a single letter is used to represent a variable, and it represents an value that is yet unknown. For example, in the equation `p = 2l + 2w`, the letters `l` and `w` represents the length and width of a rectangle. Once a person measures the length and width of the rectangle, the values can be replaced in the equation, for example, p = 2(5) + 2(2) = 10 + 4 = 14. The perimeter of the rectangle is 14. A variable therefore *represents* a value, or you can also say it is a 'place keeper' for a value. In programming, a variable can also be referred to as 'storage space' or a 'container' for a value and it is the tool used in a program to let a program remember a value. Let us look at an example where a program needs to use a variable.

In Unit 1 you created a program (a small game) where the user had to try to click on a moving sprite (the Ball2 sprite). When the user managed to click on the ball, it changed position, and a sound was played (Boing).

Now we would like to improve on this game, so that we can keep score of the number of times the ball has been hit. This is what we will need to do:



- Create a ‘container’ where the computer can put the score to ‘remember’ the number of hits (a variable).
- When the program starts, we have to place the value 0 (zero) in the container, to indicate that no hits have yet been made.
- Every time the ball is hit, we instruct the computer to increase the value in the container by one.



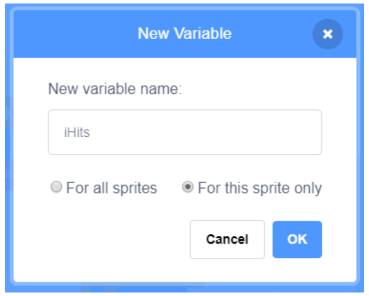
In programming languages, this ‘container’ for a value is called a *variable*. A variable has a name and can hold only one data item (for example a number, or a name) at a time. In the following activity we will show you how to add a variable to the existing program to count the number of hits.

- Variable

Activity 1: Add a variable to an existing program

Open the program you wrote in Unit 1, Activity 11.

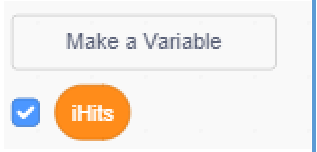
- Add a variable (a container).
  - Click on the *Variables* class in the blocks palette.
  - Choose *Make a Variable*.
  - Select the radiobutton *For this Sprite* only
  - Enter a variable name, e.g. *iHits*, and click *OK*.


New Variable

New variable name:

iHits

Note: When you write more complex programs, it will matter whether a variable is visible to all sprites, or to a specific sprite only. The difference will be explained later. In this specific program it does not matter what option you choose, as there is only one sprite. However, it is a good programming principle to assign variables to the sprite it 'belongs' to.



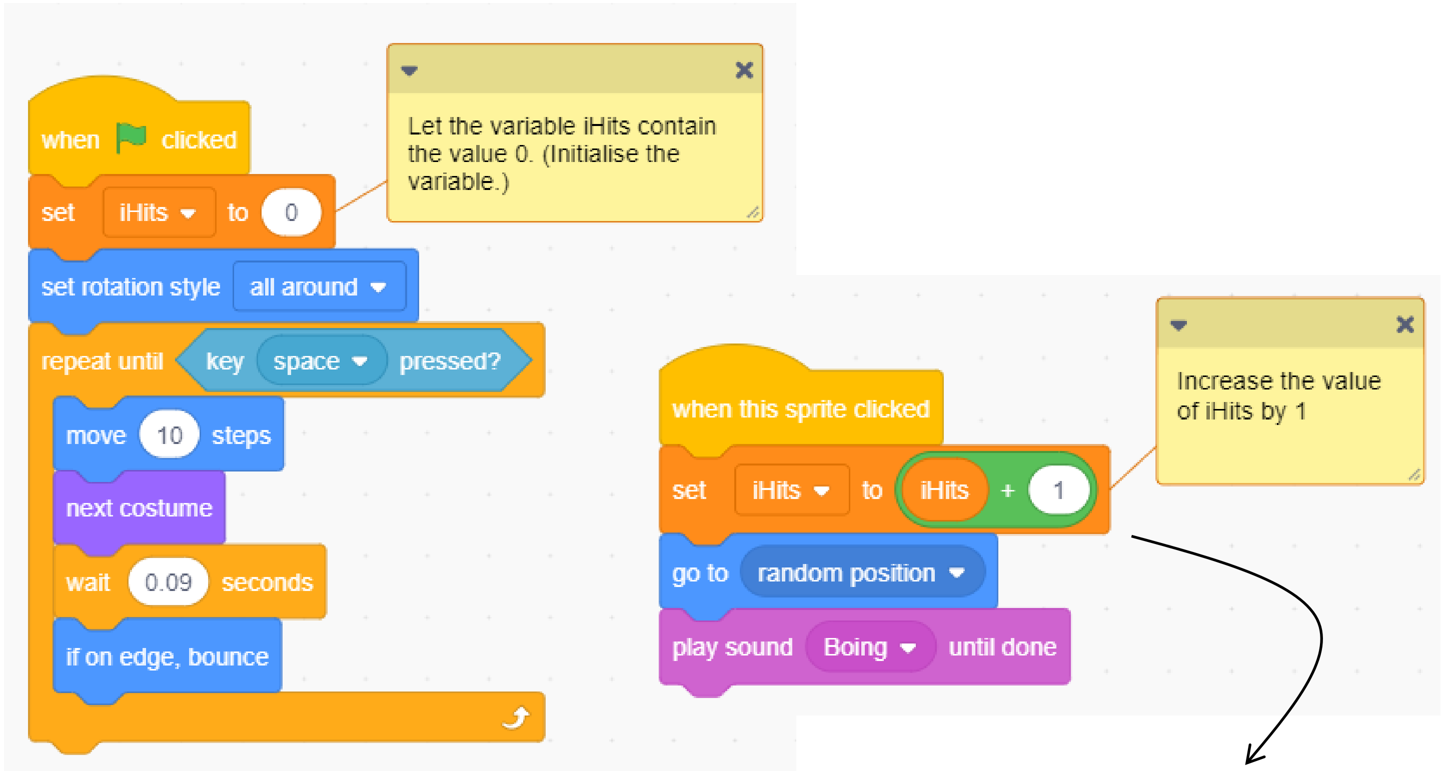
- Tick the checkbox next to the name of the variable.

- Radiobutton
- Checkbox

It is good programming practice to use meaningful names for variables, that describe both the purpose and the type of the value to be stored in them. The type of value which will be stored in this variable must be a whole number, i.e. an integer, therefore we add the letter i in front of the variable name to remind ourselves that the variable contains an integer value.

- Initialise
- Assignment statement

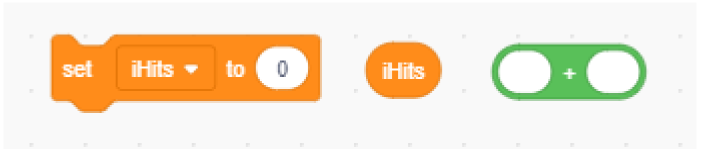
- Add code blocks as indicated (the blocks can be found in the Variables class).


```
when [flag] clicked
set iHits to 0
set rotation style all around
repeat until key space pressed?
    move 10 steps
    next costume
    wait 0.09 seconds
    if on edge, bounce
```
Let the variable iHits contain the value 0. (Initialise the variable.)

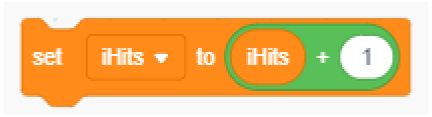
```
when this sprite clicked
set iHits to (iHits + 1)
go to random position
play sound Boing until done
```
Increase the value of iHits by 1

Readout: The Scratch terminology for the small window on the stage that displays information about a variable. The readout appears on the screen when you tick the checkbox next to the name of the variable in the Variables class.

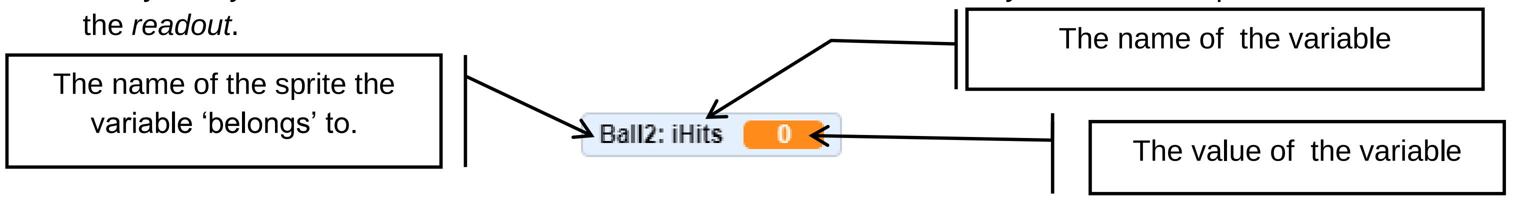
To create the instruction to increase the variable iHits by 1, look at the colours of the blocks to determine from which class to obtain the separate blocks. Add them in the scripts area.


```
set iHits to 0    iHits    ( ) + ( )
```

Now combine the different blocks, and add the number 1.


```
set iHits to (iHits + 1)
```

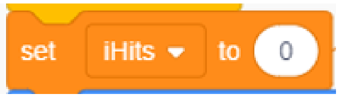
- Run the program. Notice how the variable name and the value of the variable are displayed on the stage.
- Every time you click on the ball, the value of the variable will increase by one, and the updated value will be displayed in the *readout*.


The name of the sprite the variable 'belongs' to.
Ball2: iHits
0
The name of the variable
The value of the variable

Right click on the readout. You can then select the format of the readout.

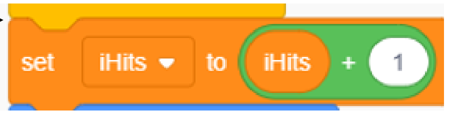
Note the following programming language concepts

- This statement is called an *assignment* statement.
- The value 0 is assigned to (placed into) the variable called iHits.
- This corresponds to an algebra equation such as x = 0. In this case iHits = 0.



```
set iHits to 0
```

- This statement is also an assignment statement.
- The current value of iHits is increased by 1. The result is 'placed back' in to the variable iHits.
- This corresponds to an algebra equation such as y = y + 1. In this case, iHits = iHits + 1.



```
set iHits to (iHits + 1)
```

Remember

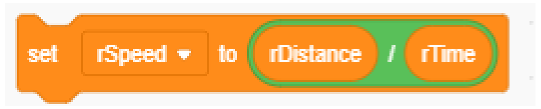
- Programming variables are given meaningful names. You can name a variable x, or y. But in longer programs with many variables you will become confused if you only use single-letter variable names. In programming we use meaningful variable names such as iNumStudents, iTestTotal, iScore instead of x, y and z.
- An *integer* is a number that is not a fraction, a whole number, such as 10; 25; 166.
- In programming languages values that have a decimal point in them, such as 12.8; 145.07823; 0.678 are called *floating point* numbers. They can also be called *real* numbers, measuring numbers or measurement numbers. In programming languages it is important to note that the value 5 is an integer, but the value 5.0 is a decimal (real) value.
- When we name variables in a program, it helps to include a reference to the type of number that will be stored in a variable. For example, **i**Hits indicates that an integer value will be stored in the the variable. **r**Avg indicates that a real number will be stored in the variable.

Assignment statement vs Equation

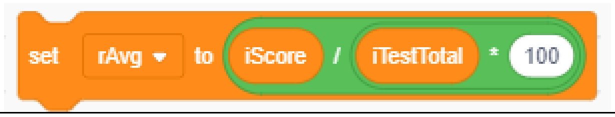
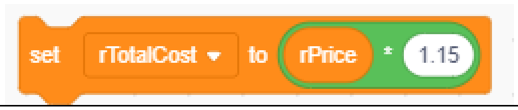
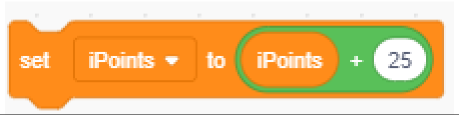
In algebra you learned when the answer of an expression is calculated, it becomes an equation. For example: if we calculate speed, we can write an equation (using meaningful variable names0:

$$speed = \frac{distance}{time}$$

In programming we use equations to determine how certain values should be calculated, and the result of the calculation on the right-hand side of the equation is assigned to a variable on the left-hand side. In Scratch we will code:

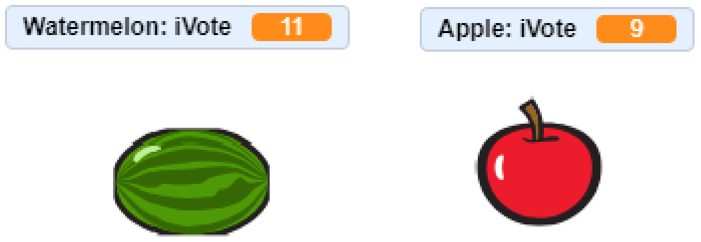

```
set rSpeed ▾ to (rDistance / rTime)
```

Here are examples of other algebra equations and their corresponding Scratch assignment statements:

$avg = \frac{score}{testTotal} * 100$	 `set rAvg ▾ to (iScore / (iTestTotal * 100))`
$totalCost = price * 1.15$	 `set rTotalCost ▾ to (rPrice * 1.15)`
$points = points + 25$	 `set iPoints ▾ to (iPoints + 25)`

Activity 2: Create variables for different sprites

- Create a program with two sprites – an apple and a watermelon.
- The cafeteria manager wants to determine which fruit he should have availble for students, so he sets up a computer at the cashier, and asks students to click on their favourite fruit.
- The program should display the number of votes for each fruit every time a student clicks on a fruit sprite.



Helpful hints:

- Create a variable for each sprite. Choose the option *For this sprite only.* The variables for the different sprites can have the same name (iVote). The readouts on the stage includes the names of the sprites, so it is clear to see whose variable each value belongs to.
- Add instructions for each sprite to initialise the variables to 0 when the green flag is clicked.

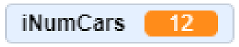
Note

- In Activity 2 you used two sprites, and each had one variable. Each had control over their own variable.

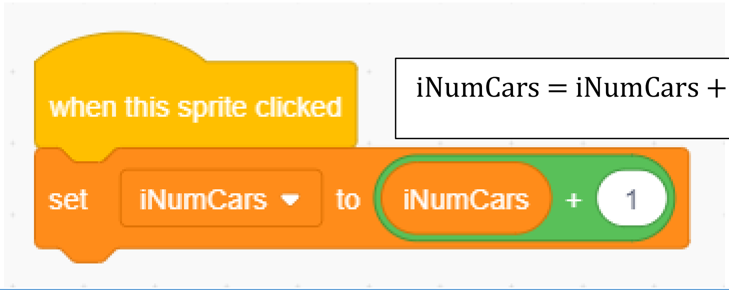
Activity 3: Worked example: Use variables visible to the stage and all sprites

- Create a program containing two sprites – the letter I and the letter O.

- This program can be used to keep track of the number of cars inside a parking lot. When a car drives into the parking lot, the user clicks on the letter I. When a car drives out, they click on the letter O.
- The total number of cars currently in the parking lot should be displayed on the stage.

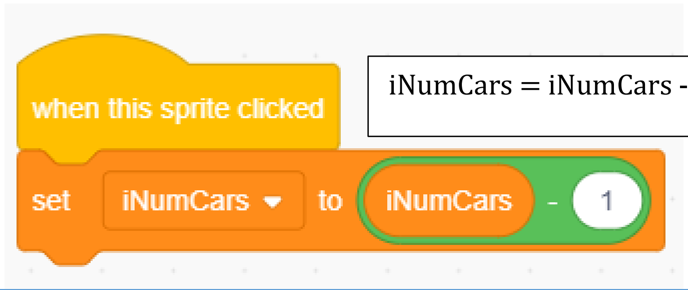


  - Click on *Variables* in the blocks palette.
  - Choose *Make a Variable*.
  - Select the radiobutton *For all sprites*.
  - Enter a variable name, e.g. *iNumCars*, and click *OK*.
  - Add a script for each sprite. When the user clicks on the letter I, the value of iNumCars should be increased by 1. When the user clicks on the letter O, the vaue of the variable iNumCars should be decreased by 1 (it is the same variable).



```
when this sprite clicked
set iNumCars to (iNumCars + 1)
```

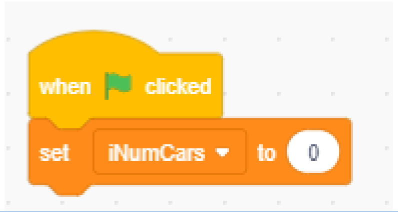
iNumCars = iNumCars + 1



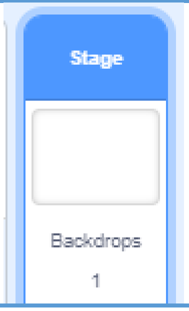
```
when this sprite clicked
set iNumCars to (iNumCars - 1)
```

iNumCars = iNumCars - 1

- Add an instruction on the stage to initialise the variable iNumcars to 0.
- Click on the Stage icon. Add code for the [when green flag clicked] event.



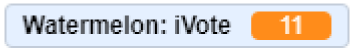
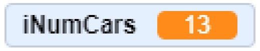
```
when 🏳 clicked
set iNumCars to 0
```



Note

- In Activity 3, you used 2 sprites, but just 1 variable (iNumCars).
- When the variable iNumCars was created, you chose the option *For all sprites*. This means all sprites can access the variable (make changes to it). This variable 'belonged' to the stage. Just like actors can see all objects on the stage, the sprites can see and change all variables on the stage.
- To initialise the variable iNumCars, it is best to add the code to the script of the stage.

The difference between For all sprites and For this sprite only

For this sprite only	**For all sprites**
Only the sprite can access the value of the variable.	All sprites on the stage have acces to the variable.
The variable has *sprite scope.*	The variable has *stage scope*.
Similar to an actor who can only read the words on his own script, not that of another actor.	Similar to an actor that can see all banners or props that is placed on the stage.
The readout on the stage displays the name of the sprite before the name and value of the variable.  Watermelon: iVote 11	The readout on the stage only displays the name and vlaue of the variable.  iNumCars 13

- Banner
- Prop

The objects used on a stage or screen by actors when they perform is called theatrical property. This has been shortened to 'props'. So a prop refers to anything movable or portable on a stage.

- Stage scope
- Sprite scope.

In the next Activity you will use 4 sprites, and 1 variable. Each sprite must have access to the variable.

Activity 4: Food nutrition – variable with stage scope

Write a program where the user should click on the food he eats every day. Every type of food has a point associated with it.

Muffin	+10
Cake	-20
Fruit salad	+25
Cheesy Puffs	-12

The starting score is 0. Every time an item is chosen, the score should be adjusted by the points for that type of food. The program should display the total score for the user.

iFoodScore 67

Remember to initialise the variable iFoodScore to 0 by placing code on the stage.

Examples of the assignment statements you will use are:

iFoodScore = iFoodScore + 10

iFoodScore = iFoodScore - 12

This variable has stage scope. Therefore no name of a sprite is displayed infront of the readout. All sprites have access to the same variable.

The only difference between this program and the previous ones are that you have more sprites, and that the variable is not just increased or decreased by 1.

Getting values from the user (user interactivity - input)

Most programs interact with the user of the program – the user needs to click somewhere on the screen or enter data (such as numeric values) on the keyboard. We will now learn how to write programs that can read values entered by a user on the keyboard, and then use that data to display a message or perform some operations.

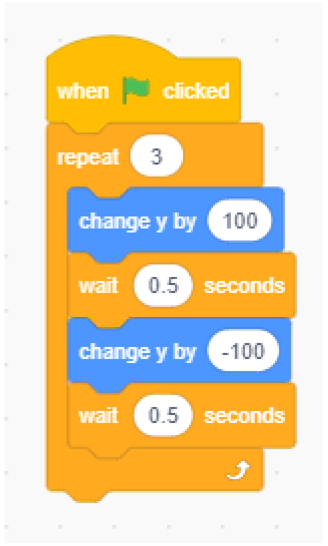
Activity 5: Write a Scratch program based on pseudocode

Write the following program in Scratch (the pseudocode is provided just to remind you again how instructions for a program can be given in a way that is not specific to a programming language).

Pseudocode

```
when the user clicks the green flag
     repeat 3 times
        go 100 steps towards top of screen
        wait 0.5 seconds
        go 100 steps towards bottom of screen
        wait 0.5 seconds
     end repeat
end
```

***Scratch* code**



```
when [flag] clicked
repeat 3
    change y by 100
    wait 0.5 seconds
    change y by -100
    wait 0.5 seconds
```

Save the program as **`Unit1 JUMP`**.

This program let the sprite jump up and down the screen 3 times. Now we want to change the program so it will ask the user how many times the sprite should jump up and down. The user will type a value on the keyboard, and push the <Enter> key. The program will then use the value entered by the user in the [repeat] block. When values are provided by the user on the keyboard, it is called *input* to the program. (A value is placed **IN** to the program).

Activity 6: Get a value from the user (input)

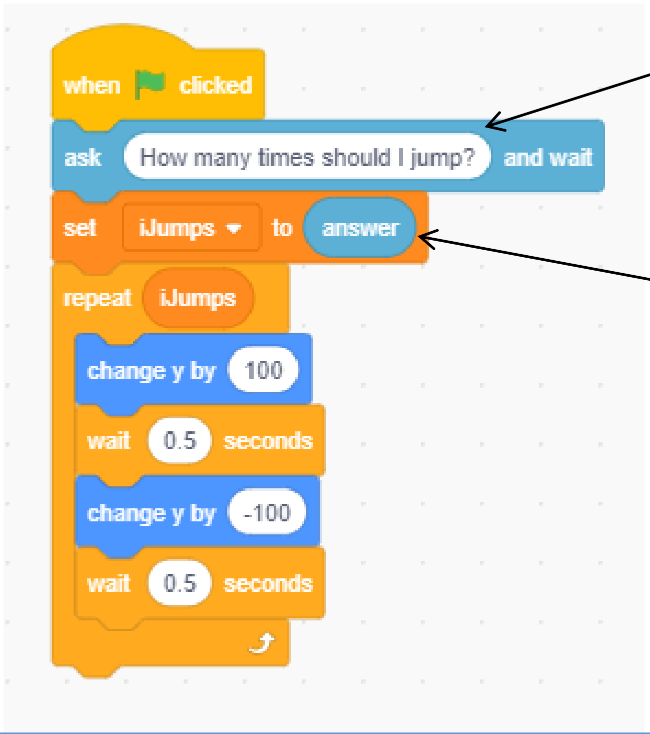
Change the program Unit 2 JUMP to ask the user how many times the sprite should jump. You need to create a variable iJumps.

Pseudocode

```
when the user clicks the green flag
      ask "How many times should I jump? "
      enter iJumps
      repeat iJumps times
            go up 100 steps
            wait 0.5 seconds
            go down 100 steps
            wait 0.5 seconds
      end repeat
end
```

Note: The number of jumps will be an integer value (the sprite cannot jump a decimal number of times – for exampl, 4.5 times). So a lower case **i** is added in front of the name of the variable.

***Scratch* code**

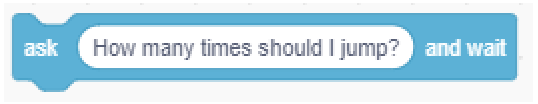
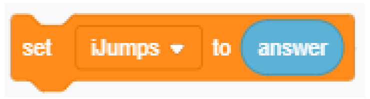


```
when [flag] clicked
ask (How many times should I jump?) and wait
set [iJumps] to (answer)
repeat (iJumps)
    change y by (100)
    wait (0.5) seconds
    change y by (-100)
    wait (0.5) seconds
```

The [ask] block is from the Sensing class

The [answer] variable can be found in the Sensing class. It is the variable where the value that is typed on the keyboard is stored automatically.

New knowledge explained

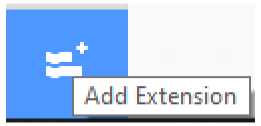
Pseudocode	**Scratch block**	**Description**
ask "How many times should I jump? "		The [ask] block (in the *Sensing* group) prompts the user to enter the data requested by the programmer using the keyboard.
enter iJumps	 The content of the variable called answer, is on the right hand side of the block, and it is transferred to the variable sName - on the left-hand side of the block.	• Scratch automatically puts the value that is entered on the keyboard into its own variable, *answer*. • The [set] block (from the *Variables* class) then transfers the data from the variable *answer* to the variable iJumps (declared by the programmer.)

Activity 7: Obtain two input values

Create a Scratch program using the following pseudocode. Choose a smaller sprite such as a ball. Note that you have to create two variables.

```
when the user clicks on the green flag
      clear all previous graphics effects
      lift the pen up
      move to position (-160, 160)
      look to the right of the screen
      put the pen down
      ask "How many steps should I take to draw a square (between 100 and 300)?"
      enter iSteps
```

Code blocks to manipulate a pen can be found in the *Pen* class. You need to click on the *Add extension* icon to make the Pen class visible

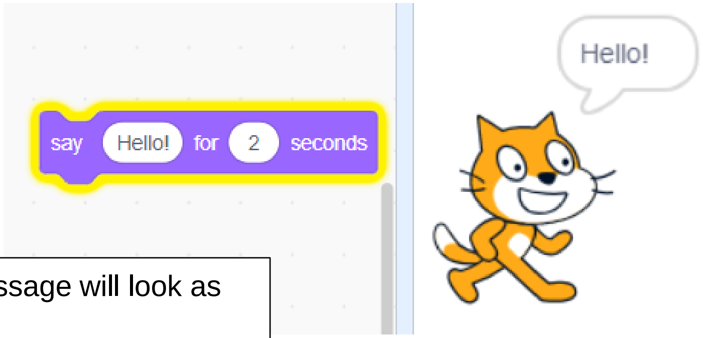


```
      ask "How thick must the pen be (between 1 and 10)?"
      enter iPen
      set pen size to iPen
      repeat 4 times
            move iSteps steps
            turn 90° clockwise
            wait 1 second
      endRepeat
end
```

Displaying messages on the screen (output)

When a computer program displays something on the screen, it is called **output** of the program. In Unit 1 you used the [say] block in many programs to display words in a speech bubble next to a sprite.

The [say] block is used to create output of a program.



In this example, the word 'Hello!' is the output of the program.

The pseudocode to indicate this type of message will look as follows:

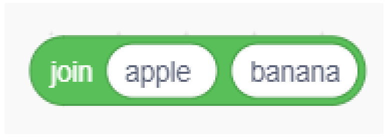
```
display "Hello!"
```

The output in this example is a word. In programming terminology a word or a sentence consisting of alphabetical characters and/or digits and/or other special characters (e.g. $ or *) is referred to as a *string* or *text*.

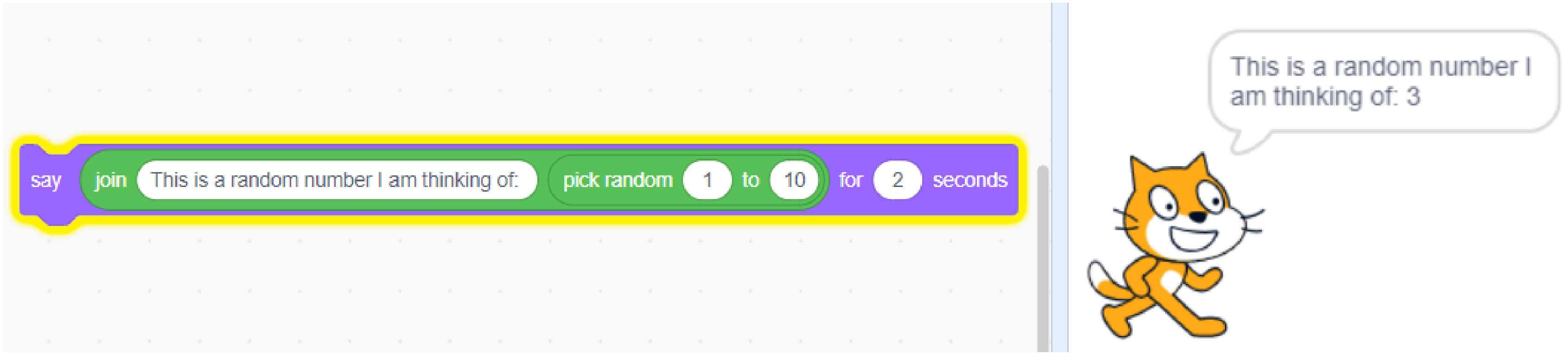
In pseudocode and most programming languages' code, double quotes are used to indicate that a value is a string, for example, "Hallo!" is a string. These are more examples of strings:

```
"I am dancing"
"This is gr8!"
'PO Box 58234"
```

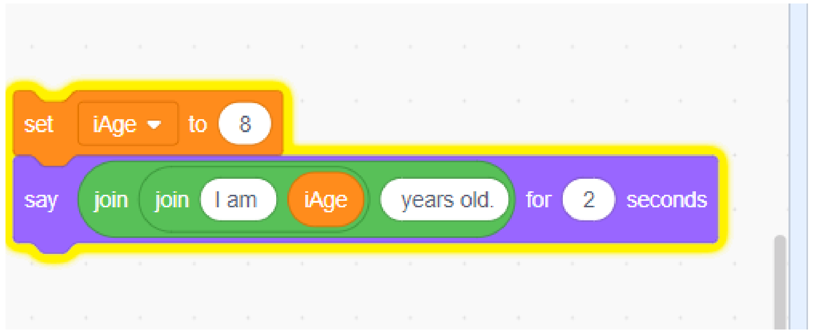
You can combine blocks to create output using a [join] block from the operators class. The [join] block has two editboxes where you can enter sentences or variable names to display.



These are examples of using [say], [join] and other blocks to create output:


say join This is a random number I am thinking of: pick random 1 to 10 for 2 seconds
This is a random number I am thinking of: 3

You can use multiple [join] blocks to concatenate (add together) different components to form one output message.



```
set iAge to 8
say join (join "I am" iAge) "years old." for 2 seconds
```

I am 8 years old.

The pseudocode to indicate this code will look as follows:

```
iAge = 8
display "I am " + iAge + " years old"
```

The + symbol indicates that different components must be joined to form the output. These components are:

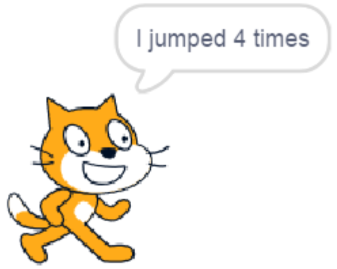
`"I am "`	+	`iAge`	+	`" years old"`
A string	+	A variable	+	A string

Activity 8: Display output

Add code to the program Unit 2 JUMP so the sprite will display a message to report how many times it jumped. For example:

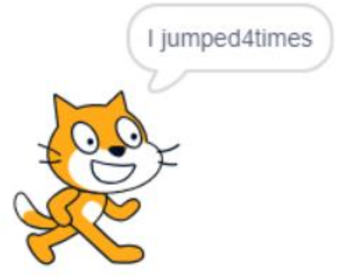
The pseudocode to indicate this type of message will look as follows:

```
display "I jumped " + iJumps + " times"
```



Note the spaces that you will have to leave at the end of the first string, and the beginning of the second string.

If you do not leave any spaces in the output strings, the output will ook like this:

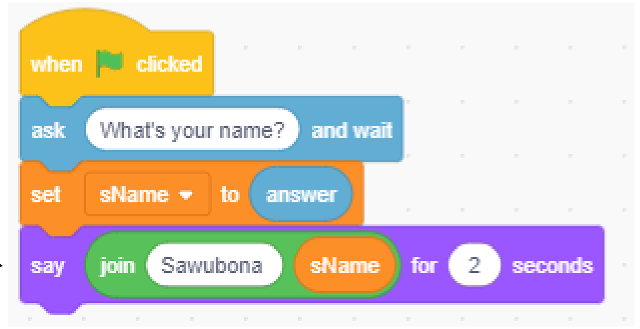


Look at the following algorithm in pseudocode and the corresponding *Scratch* code.

Pseudocode

```
when the user clicks the green flag
      ask "What's your name? "
      enter sName
      display "Sawubona " + sName
end
```

***Scratch* code**


when clicked
ask What's your name? and wait
set sName to answer
say join Sawubona sName for 2 seconds

Note the following

- Each line of the pseudocode is translated to a programming instruction (program block).
- The name of a preson consists of letters, therefore it is a string, and we add the letter 's' to the variable name to remind us that the variable will contain a string. The name of the variable that will contain the value entered by the user o n the keyboard is sName.
- To display a space between the word *Sawubona* and the value of sName, enter a space after the word *Sawubona*. This can also be seen in the pseudocode example, where a space was typed before the apostrophe closing the text string – "Sawubona ".

Activity 9: Display name and age

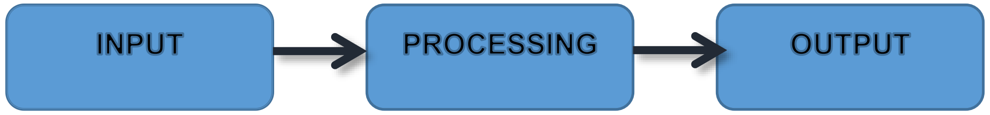
Write pseudocode for a program where the sprite will ask a person’s name and age, and display a sentence such as: 'Good day Katlego, you are 16 years old.' Write a Scratch program based on the pseudocode.

- Interface
- Comparisons
- Retrieve

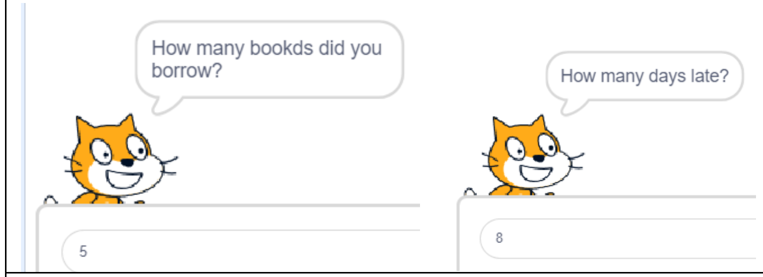
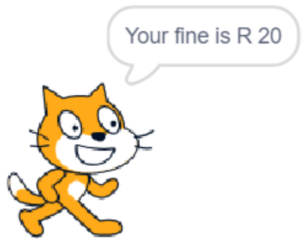
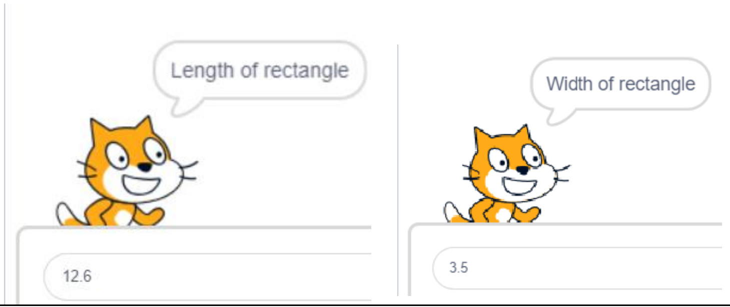
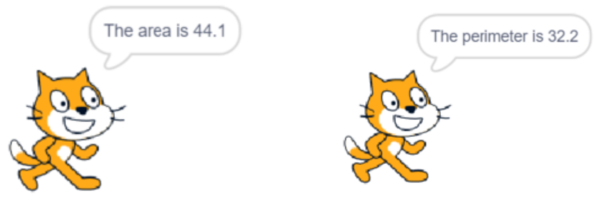
Writing basic programs

The programs we have written up till now have been very plain, designed to introduce you to the *Scratch* interface and basic programming language terminology. However, the actual usefulness of computers lies in their ability to perform calculations and process data quickly and accurately. The aim of learning to program is learning how to instruct the computer to process data and perform calculations.

Before attempting to write code to solve problems, we need to understand how computers work. In short, computers receive input, process the input and give output.



Here are some examples of input and output.

Input Values entered by the user of the program.	**Processing**	**Output** Results / answers displayed by the computer.
 How many bookds did you borrow? 5 How many days late? 8	Calculations done by the computer. As the user of the program, you do not have to kow how the calcuations are done.	
		

For computers to accomplish a task they

- receive input (from various sources, e.g. keyboard, mouse, storage media)
- perform calculations (usually mathematical in nature)
- make comparisons (and then perform certain actions as a result of this)
- give output (to various devices such as the screen, storage media, printer)
- store and retrieve data
- So the first thing we have to learn in programming is to *identify input and output,* and *decide on the processing that will be needed.* We will now show you some examples on how to plan and write programs which involve calculations.

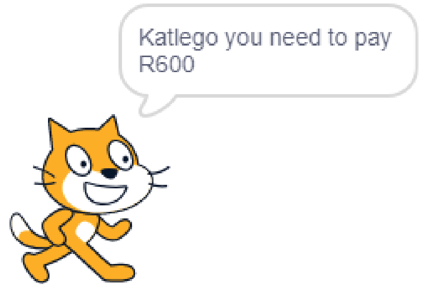
Worked example: Cost of tickets

TUT students are selling tickets for a Cassper Nyovest concert. The price of a ticket is R200. You need a program that will read the name of the student who wants to buy a ticket as well as the number of tickets the student wants to buy (the name and number is the input to the program – the information to be supplied by the user). The computer must then work out the amount to be paid and display the result with an appropriate message. For example, 'Katlego you need to pay R600'. The result to be displayed is the output of the program.

Solve the problem yourself

Before you can write instructions for the computer to do calculations, you need to analyse the problem, and be able to solve the problem yourself.

Plan Constants, Input, Processing and Output

CONSTANT VALUE(S) and INPUT	PROCESSING	OUTPUT
Step **2**: What values do you know that will not change during the program? *One ticket's cost is R200* Step **3**: What data values do you need? *The name of the student.* *The number of tickets bought.* An *assignment statement*. Assign the value calculated on the right-hand side of the = sign to the variable on the left	Step **4**: If you assume you know the input value(s), how will you calculate the answer? *Let us pretend the input value number of tickets bought is 3. Then we would calculate the total as:* *3 * 200 = 600* *Now you can translate the mathematical calculation to an assignment statement (a formula):*	Step **1**: What do you need to calculate and/or display?  *sName (The name of the student.)* *rTotal (The total cost of the tickets.)*

Constant value (from step 2) Cost of one ticket **(***ONETICKET=200)* **Input (from step 3)** Name of the learner (sName) Number of tickets (iNumTickets)	**Processing** `rTotalCost = iNumTickets * ONETICKET`	**Output** Name of the learner (sName) Amount payable (rTotalCost)

This table is called an Constant-Input-Processing-Ouput (CIPO) table. It is one of the tools you can use to develop an algorithm when you plan a program.

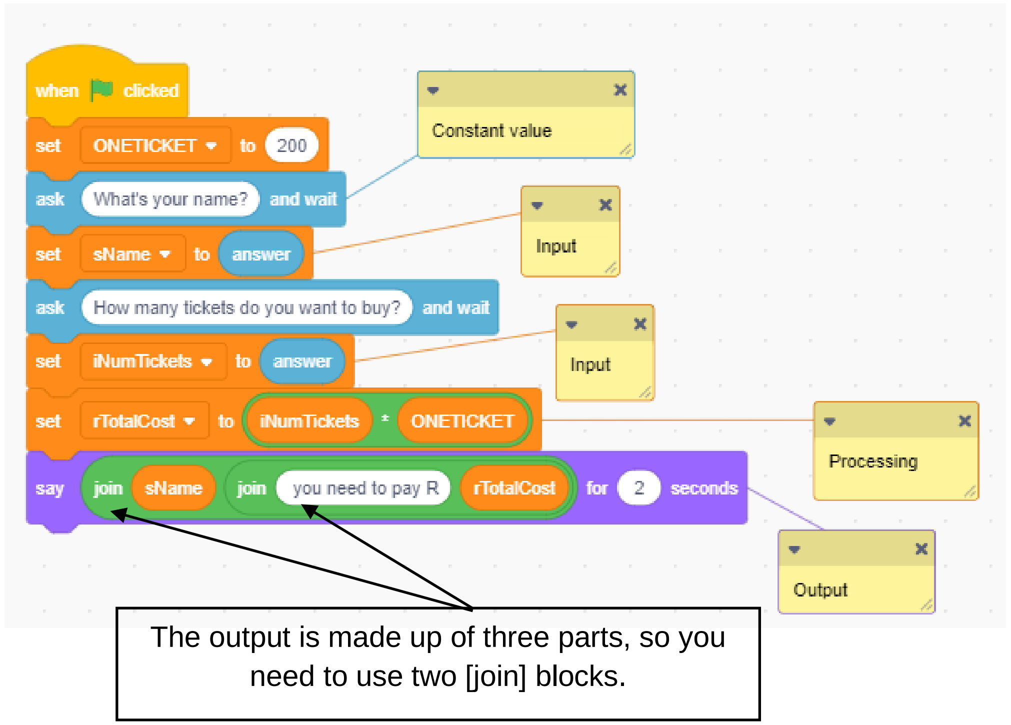
Write an algorithm for the program.

```
The complete algorithm in pseudocode is as follows:
when the user clicks the green flag
    ONETICKET = 200                                    Constant
    ask "What is your name? "
    enter sName
    ask "How many tickets do you want to buy? "        Input
    enter iNumTickets

    rTotalCost = iNumTickets * 2                       Processing

    display sName + " you need to pay R" + rTotalCos   Output
end
```

The string created using the [join] blocks will be displayed in the speech bubble close to the sprite. You can also use the readouts for variables to display the values.

You need to display the letter R in front of the amount, so that the person who uses the program can see the amount is a currency in Rand.

Translate the algorithm to programming code.

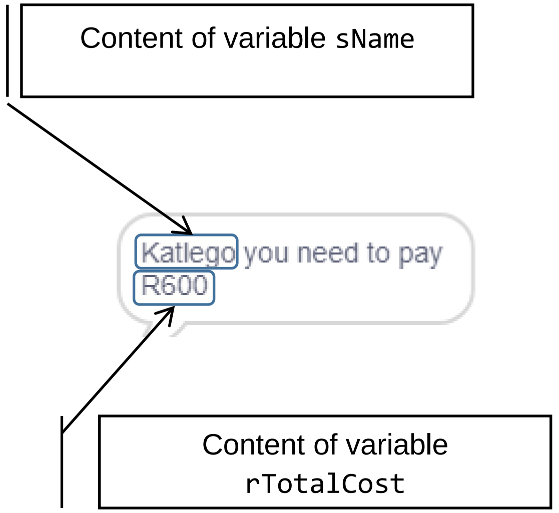


```
when [flag] clicked
set ONETICKET to 200
ask What's your name? and wait
set sName to answer
ask How many tickets do you want to buy? and wait
set iNumTickets to answer
set rTotalCost to (iNumTickets * ONETICKET)
say join sName join you need to pay R rTotalCost for 2 seconds
```

The output is made up of three parts, so you need to use two [join] blocks.

NOTE

Always use the output that should appear on the screen to identify your output variables' names.



Content of variable sName
Katlego you need to pay R600
Content of variable rTotalCost

It is good practice to use capital letters only for the name of values that will not change during the run of the program. For example, in the program calculating the cost of the ticket, it will stay R200 per person. If you want to change the value to R250, you can see easily where the fixed values (constants) in your program is when you use capital letters in the names.

Activity 10: Cost of tickets

Watch the video to see how to create the output using the say and join blocks, then write the program to calculate the cost of all tickets in Scratch.

Run the program a number of times. Use different names and number of tickets each time. See how the output changes because the program uses the value of the variables you supplied.	4_JoinBlocksForOutput

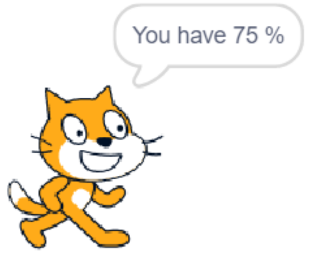
Worked example: Calculate percentage

A student needs a program to calculate the percentage for a test. The student's score, and the total marks for the test needs to be entered.

Solve the problem yourself

Again you need to analyse the problem, and be able to solve the problem yourself.

Plan Constants, Input, Processing and Output

CONSTANT VALUE(S) and INPUT	PROCESSING	OUTPUT
Step **2**: What values do you know that will not change during the program? *There are no fixed values* Step **3**: What data values do you need? *The score of the student.* *The total of the test.*	Step **4**: If you assume you know the input value(s), how will you calculate the answer? *Let us pretend the student scored 45 out of 60. Then we would calculate the percentage as follows:* *45/60*100 = 75* *You can write it as a formula:*	Step **1**: What do you need to calculate and/or display? *The percentage for the test.* 
Input Score of the student (iScore) Total of the test (iTestTotal)	**Processing** `rPercentage = iScore/iTestTotal * 100`	**Output** The percentage of the test (rPercentage)

Write an algorithm for the program.

The complete algorithm in pseudocode is as follows:

```
when the user clicks the green flag
      // Input
      ask "What is your score? "
      enter iScore
      ask "What is the total for the test? "
      enter iTestTotal
      // Processing
      rPercentage = iScore/iTestTotal * 100
      // Output
      display "You have " + rPercentage + "%"
end
```

Comments.

Translate the algorithm to programming code.

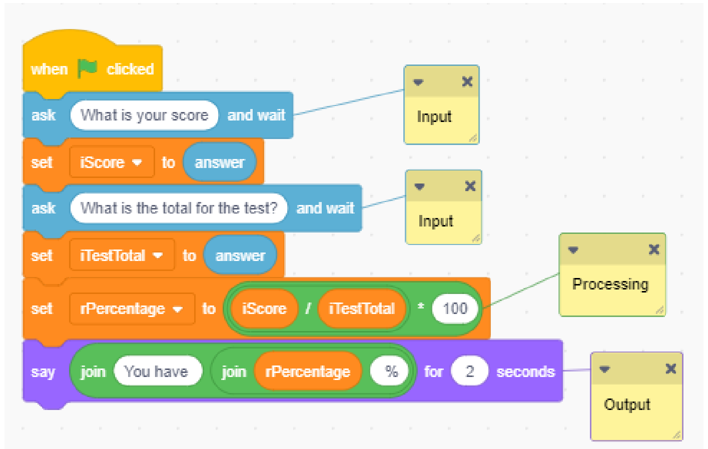
The two forward slashes //, is used to indicate comments in a program. Anything you write after the //, provides more information about the program.

- Test data

You can calculate the percentage of 45/60. It is 75. When you run your program, you can use the value 45 for iScore, and 60 for iTestTotal. Your program should display 75%. The values 45 and 60 is then called *test data* for your program.

NOTE

Your program should work for ANY score and total, not just 45 and 60. That is why you have to use variables in a program.


```
when [flag] clicked
ask (What is your score) and wait
set [iScore] to (answer)
ask (What is the total for the test?) and wait
set [iTestTotal] to (answer)
set [rPercentage] to ((iScore) / (iTestTotal) * (100))
say (join (You have) (join (rPercentage) (%))) for (2) seconds
```

Activity 11: Calculate percentage

- Watch the video to see how to create the instructions to do calculations using operator blocks.
- Write the program to calculate the percentage when a test score and total is read from the keyboard.
- Run the program a number of times, using different values for the score and test total.

5_ CalculationsUsingOperatorBlocks

Scratch will display all the decimal values for the answer. E.g. 44/60 = 73.333333333%. When you learn to program using Java, you will learn how to change the value of a variable to show a certain number of decimal values.

Order of precedence in calculations

We already know how to perform calculations and apply mathematical rules. For example,

3 + 4 * 5 means 3 + (4 * 5) = 3 + 20 = 23.

Most programming languages have built-in ‘knowledge’ of the order of precedence in calculations; for example, multiplication and division are performed before addition and

In Computational Mathematics you learned the order of precedence of arithmetic operations:

Parentheses ()
Exponents ^
Multiply Divide MOD (* / \ %)
(operator on left side first)
Add Subtract (+ -)
(operator on left side first)

NOT the same answer!

subtraction. If the following calculation is given to a computer, it will calculate 12/4 first.

3 + 8 + 12/4 = 3 + 8 + 3 = 14

When you want addition or subtraction to happen first in a calculation, you need to use parentheses to ensure that is does happen first. For example, when calculating the average of three numbers, it should be done as follows::

`(70 + 65 + 81) / 3 = 216 / 3 = 72`

as pseudocode we can write `rAvg = (iMark1 + iMark2 + iMark3) / 3`

However, if the calculation is performed as follows (no brackets):

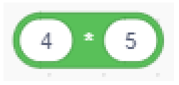
`avg = iMark1 + iMark2 + iMark3 / 3` it will be interpreted as `70 + 65 + 81 / 3 = 70 + 65 + 27 = 162`

Division is done before plus.

When writing a program, the programmer has to apply mathematical principles correctly when programming calculations – such as using parentheses – the programming language has no way of ‘knowing’ that parentheses should have been used to calculate the average. It only performs the instruction as it was provided.

In *Scratch*, the effect of parentheses and the order of precedence in calculations is achieved by *stacking* various program blocks from the *Operators* group. In other programming language the calculations are typed on one line, and the language will automatically do the calculations in the correct way (following rules for order of operations.

Here are the steps to program the calculation 3 + 4 * 5 in Scratch:

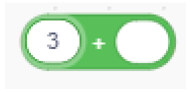


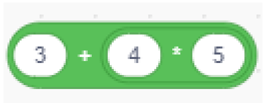
1. The multiplication has to be completed first. So drag a multiplication block from the Operators class to the scripst area, and enter the values 4 and 5.
2. Drag an addition block onto the scripts area. Enter the value 3.
3. Now plug the multiplication operator into the open slot on the right side of the addition

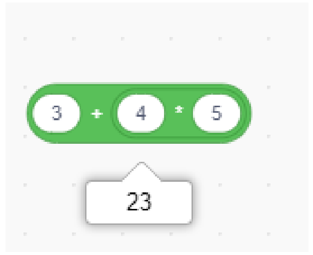
operator.

Notice how the multiplication block appears to be raised higher. It is at the top of the stack.

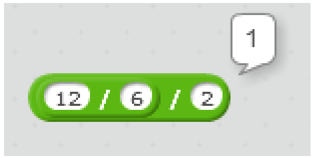
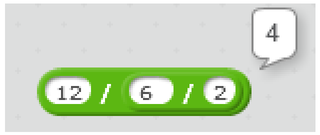
Scratch will perform the calculation at the top of the stack (i.e. the multiplication) first.

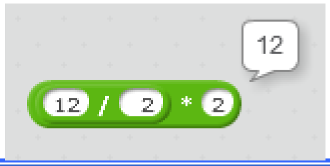
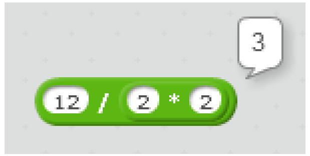






Here are more examples of the way stacking should be used to ensure the correct order of precedence:

12 ÷ 6 ÷ 2 $=\frac{{}^{12}/_{6}}{2}$ = 12/6/2  This should be programmed as (12/6)/2, since the rules for the order of precedence) indicate that division is carried out from left to right.	12 ÷ (6 ÷ 2) $=\frac{12}{{}^{6}/_{2}}$ = 12/(6/2) 

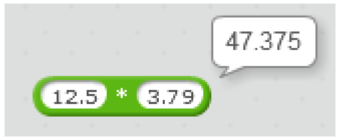
`12 ÷ 2 x 2`
$=\frac{12}{2}\,x\,2$
`= 12 / 2 * 2`
This should be programmed as (12/2)*2, since division and multiplication are on the same ‘level’ in the order of precedence, and should be performed from left to right.

	`12 ÷ (2 x 2)`
	$=\frac{12}{2*2}$
	`= 12 / (2 * 2)`
	

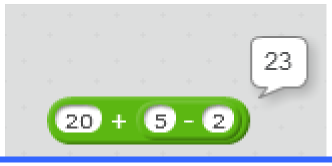
Activity 12: Practise stacking to effect order of precedence

Use mathematical operator blocks to calculate the following:

1. Add 5 to 20.
2. You scored 45 out of 60 for a test. Calculate your percentage.
3. Calculate the average of the three numbers 68, 89 and 77.
4. Calculate (3 + 2) / (1 + 1).
5. Determine the speed of a car that travels 14 km in 8 minutes, in km/h.
6. Increase the value of 100 by 15%.
7. Calculate 15% of 100.
8. Decrease the value of 100 by 15%.

If you want to test a calculation quickly, you do not need to write a complete program using a sprite and program blocks. You can use a mathematical operator block only, enter values in the slots, and left-click on the block. A speech bubble will appear containing the answer





Activity 13: VAT and total amount

Use the template provided below to plan a solution, and write an algorithm as well as a Scratch program for the following problem:

Read the amount a customer spent at the bookshop. The program must calculate and display the amount of VAT the customer should pay, as well as the total amount (original + VAT) to be paid.

Plan Constants, Input, Processing and Output

CONSTANT VALUE(S) and INPUT	PROCESSING	OUTPUT
Step **2**: What values do you know that will not change during the program?	Step **4**: If you assume you know the input value(s), how will you calculate the answer?	Step **1**: What do you need to calculate and/or display?

VAT is 15% of the amount spent. Step **3**: What data values do you need?	*Let us pretend the customer spent R245.* Calculate the amount VAT Calculate the total to be paid.	*The amount VAT to be paid.* *The total to be paid..*

Input	**Processing**
	
	`rVAT =`
	`rTotalToPay =`
		Output

Write an algorithm for the program.

Complete the algorithm in pseudocode:

```
when the user clicks the green flag
      //Constant value
      VAT_PERC = 0.15
      // Input
      ask "What is the amount spent? "
      enter rSpent
      // Processing
      rVat =
      rTotalPay =
      // Output
      display
      display
end
```

Translate the algorithm to programming code.

Test data: Use the following values to test your program.

rSpent	*rVat*	*rTotalPay*
100	15	115
250.88	37.632	1348.512

NOTE

- Every time you run the program, you can use a different value for the amount spent. That is why the input and output values are called variables.
- You should not program the values of the test data e.g. 100 or 250.88 in the code of the program.

All decimal values will be displayed, not just 2 as we would prefer when displaying a monetary value. You will learn how to display only 2 decimal values for values when you program in Java.

- Monetary value

Activity 14: Pocket money

Ask the user's age. Calculate the pocket money to be paid. Pocket money per month is calculated as R20 for each year of the age. So a person who is 5 years old, will receive R100 pocket money per year. Let year be an integer value.

Create an IPO-table, then complete the algorithm and write a Scratch proram to implement the algorithm.

Activity 15: Calculate average of 3 values

Enter the name of a student and the student's marks (as percentages) for three class tests written this term. Determine the average of the three marks. Display the student's name and average as output.

Create an IPO-table, then complete the algorithm and write a Scratch program to implement the algorithm.

Activity 16: Calculate age this year

Ask the user for the current year and the user's year of birth, and then calculate and tell the user how old he or she will be in the current year.

Create an IPO-table, then complete the algorithm and write a Scratch program to implement the algorithm.

Activity 17: Calculate perimeter and area

A builder wants to calculate the perimeter and surface area of the floor of a room. Applying your knowledge of Mathematics, you know that the length and width of a rectangular area are needed to calculate the area and perimeter. The table below shows the input and output required as well as the processing of the data.

Input	Processing	Output

`The length of the room (rLength)`
`The width of the room (rWidth)`
	`rPerimeter = 2 x rLength + 2* rWidth`
	`rArea = rLength *rWidth`
		`The perimeter (rPerimeter)`
		`The area (rArea)`

Write an algorithm to obtain data, perform calculations and display the output.

Write a Scratch program to implement the algorithm.

- Manipulate
- Decimal value

Data types of variables

All programming languages can manipulate different types of data. For example, you have already used

- integer values (whole numbers, negative or positive e.g. -5; 8; 350)
- real values (decimal values, e.g. 3.44; -6.7687686)
- strings (text values, e.g. a name "James ", or an address "2 College Road ", or a cell phone number "0845561289 ". You cannot do a numerical calculation with a string value.

5_Different DataTypes

There are more types of data, and we will learn to use them later in this module.

- Declare a variable

Scratch-specific programming principle

Most programming languages require you to indicate the data type of a variable when the variable is declared. *Scratch* does not require this. Instead, the data type is determined by *Scratch,* based on the operations and calculations performed on the variable.

Monetary values always have a decimal value – even if the amount is R 5.00 and the decimal part of the value is 0, you will want to display 2 decimal values, e.g. R5.00, or R6.70. Note: The 'R' character is not part of the value the variable contains. As the programmer you always have to display the letter 'R' in output values in Scratch.

Naming conventions for variables

Scratch does not really have rules about variable names. However, most programming languages do have strict rules about this, and may return an error if the rules are disregarded. To prevent you from becoming confused later when programming in another language, we suggest that you adhere to the following rules when naming variables:

- The first character of a variable name should be an alphabetical character (not a number or symbol), and should be an indication of the type of value that will be stored in the variable. For example: iCount (an integer value), rAverage (a real value), sName (a string).
- Do not use spaces in variable names. For example, instead of 'Term Mark', rather use the name TermMark or Term_Mark.

- Do not use non-alphabetic characters like ‘/’ and ‘*’. Only the underscore character ‘_’ is allowed.
- The name of the variable should describe its content. For example sName or iCount or iMark.
- Use a capital letter for the first character of each word that make up the variable name, e.g. sCellNumber, sDateOfBirth.

Activity 18: Variable names

Indicate whether the following variable names are good choices or not. Provide a reason when it is not a good choice, and provide a better alternative.

a) iPhoneNumber
b) sISBN
c) sUrnames
d) rNum2
e) sSum
f) sName&Surname
g) Endangered species
h) rTo
i) rAmountPayable
j) tShops
k) 18holeCourseName

Activity 19: Suitable variable names

Choose a suitable variable name for each of the following scenarios. The variable name should reflect the type of the data.

a) The name of one of your subjects.
b) Test marks.
c) The average on your report card.

d) To store ID numbers.
e) To indicate the season.
f) The variables you will need to calculate the area of a rectangle, as well as the variable where you will store the answer.
g) To store a password.
h) Two variables, inde for your grade, and one for the number of learners in the grade.
i) You want to calculate your budget. Which variables will you need to store your income, expenses, and the remainder of your money.

Activity 20: Km walked

The distance between a student's room and the campus is 500m. The student walks to campus and back home for 5 days. How many kilometers does the student walk each week?

Do the calculation using a pocket calculator and writing down all the calculations.

Refer to your calculations, and complete the following algorithm:

```
when the user clicks on the green flag
      ask "How far is your room from the campus (in m)? "
      enter rDist
      ask "How many days do you walk to campus? "
      enter numDays



      display "You have walked ", rKm, " km this week"
end
```

Complete the IPO-table for this solution

Input	Processing	Output

Write a Scratch program to implement the algorithm.

Use the following test data to determine whether your program is correct.

Distance in meter	Number of days	Km walked
500	2	2
850	5	8.5
1300	4	10.4

Activity 21: Library fine

A library charges a fine of 50c per day per book if a book is returned late.

A student has 4 books and hand them in 5 days late. How much is the fine (in Rand)?	A student has 9 books and hand them in 4 days late. How much is the fine (in Rand)?	Write a formula in pseudocode to calculate the fine if *x* books are *y* days late.

Write an algorithm using valid variable names to read the number of books and the number of days they have been returned late, then calculate and display the fine in Rand.

Write a Scratch program to implement the algorithm.

Activity 22: Buy candy

A 100g of candy costs R6.50. A program is needed where the shopkeeper can enter the weight (in gram) of the candy a customer buys, and then the program will display the amount owed by the customer.

In Mathematics you calculated it as follows:

$$\begin{tikzcd} 100g \arrow[dr, dash] & R6.50 \arrow[dl, dash] \\ rCandyWeight & rAmount \end{tikzcd}$$

Cross multiply (we do not use variables such as x and y, rather variable names that describes the unknown value).

$$100 * rAmount = 6.5 * rCandyWeight$$

$$\Rightarrow rAmount = \frac{6.5 * rCandyWeight}{100}$$

Develop an algortihm to read the weight of the candy bought, then calcuate and display the amount to be paid.

Write a Scratch program to implement the algorithm.

Use the test data provided to test your program.

Candyweight	**Amount**
350	R22.75
120	R7.80

Activity 23: Apple profit

A bag with 10 apples costs R30. If you sell each apple for R5, how much profit will you make? (Do the calculation here.)

A bag with 22 apples costs R50. If you sell each apple for R1.50, how much profit will you make? (Do the calculation here.)

Write an assignment statement in pseudocode to calculate profit using the following variable names:

iNumApples *//The number of apples in a bag*

rProfit *//Total profit when all the apples are sold* rcostBag *//Cost for the bag of apples*

rSellOne *// The amount you will charge for one apple*

Create an algorithm in pseudo code so a program will read how many apples are in a bag, what is the cost of the bag, how much will you charge for one apple. The program should calculate and display the total amount of profit when all the apples are sold.

Activity 24: Calculate phone call cost

Use the template provided below to plan a solution, and write an algorithm as well as a Scratch program for the following problem:

The first three minutes of a phone call costs 50c per minute. The remaining minutes (everything more than 3 minutes), costs 15c per minute. Write a program to calculate the total cost of a phone call when the duration of the call is provided. Assume the user will never enter a value less than 3.

You can draw a diagram such as the following one to indicate how the price of a phone call will be calculated.

first 3 minutes	4th minute	5th minute	6th minute	. . . . . .
50^c per minute	15^c	15^c	15^c	. . . . . .

Plan Input Processing and Output

Step **2**: What values must you use? *First 3 minutes cost 50c per minute* *Remaining minutes cost 15c per minute.* Step **3**: What information do you need? (Input)	Step **4**: If you use a test value, how will you calculate the answer?	Step **1**: What do you need to calculate and/or display? (Output) *The total cost of the phone call.*

	Let us say the call was for 10 minutes What is the cost of the first three minutes? How many minutes were more than 3? What is the cost for those minutes above 3 min? Write a formula to calculate the total cost.	

Input	**Processing**
	`rFirstCost = 3 * 0.5`
	`iRemMin =`
	`rCostRem =`
	`rTotalCost =`
		Output

Write an algorithm for the program.

Complete the algorithm in pseudocode:

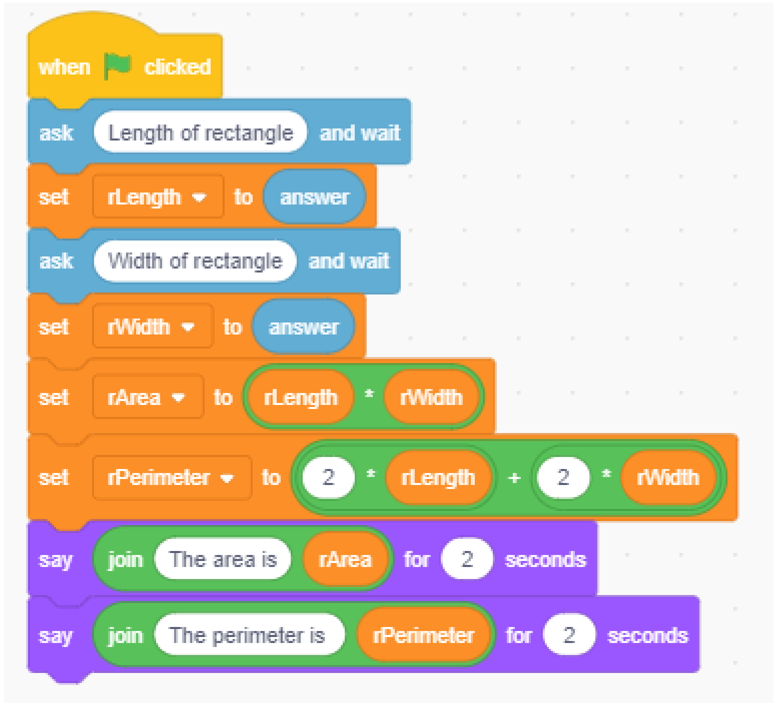
```
when the user clicks the green flag
      //Constant values


      // Input
      ask "What is the duration of the call? "
      enter iDuration
      // Processing
```

```
    // Output
    display
end
```

Translate the algorithm to programming code.

Multiple lines of output

Worked example: Output to a List

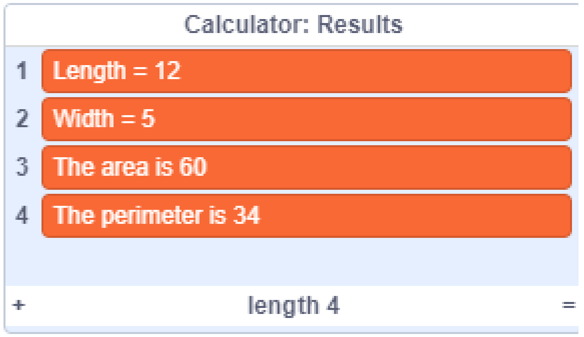


```
when [flag] clicked
ask (Length of rectangle) and wait
set rLength to (answer)
ask (Width of rectangle) and wait
set rWidth to (answer)
set rArea to ((rLength) * (rWidth))
set rPerimeter to (((2) * (rLength)) + ((2) * (rWidth)))
say (join (The area is) (rArea)) for (2) seconds
say (join (The perimeter is) (rPerimeter)) for (2) seconds
```

In a previous activity, you had to calculate the perimeter and area of a rectangle. This was the code for the program:

The way the output is displayed, is not really effective, since each value is displayed for only 2 seconds, then it disappears. It would be better to use a structure where all the output lines will be visible. In Scratch we can use a structure called a List. It can be found in the Variables class. A List is one structure that consists of many lines – each line contains one value. When we use a List as ouput structure, each line will contain a string. The output on the screen will look as follows:

- List
- String



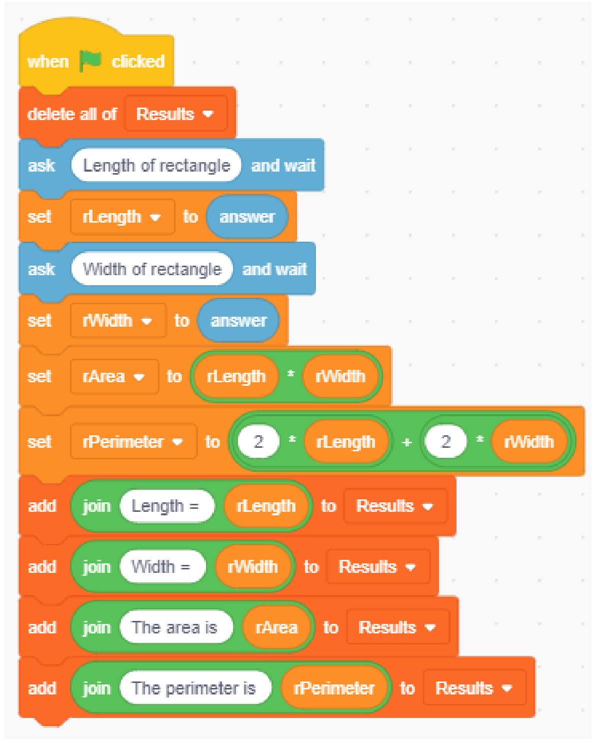
```
Calculator: Results
1 Length = 12
2 Width = 5
3 The area is 60
4 The perimeter is 34
+          length 4          =
```

- The name of the Sprite in this program is *Calculator*.
- The name of the List is *Results*.
- There are four lines of output (the number of lines is indicated at the bottom of the List – *length 4*).

The algorithm can be written as follows:

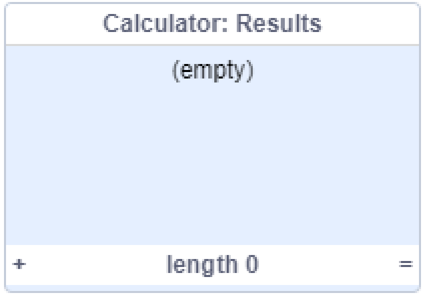
```
when the user clicks the green flag
      // Input
      ask "Length of rectangle? "
      enter rLength
      ask "Width of rectangle? "
      enter rWidth
      // Processing
      rArea = rLength * rWidth
      rPerimeter = 2 * rLength + 2 * rWidth
      // Output to a List
      addToList "Length = " + rLength
      addToList "Width = " + rWidth
      addToList "The area is: " + rArea
      addToList "The perimeter is: " + rPerimeter
end
```

The program will look as follows:



```
when [flag] clicked
delete all of [Results v]
ask (Length of rectangle) and wait
set [rLength v] to (answer)
ask (Width of rectangle) and wait
set [rWidth v] to (answer)
set [rArea v] to ((rLength) * (rWidth))
set [rPerimeter v] to (((2) * (rLength)) + ((2) * (rWidth)))
add (join (Length =) (rLength)) to [Results v]
add (join (Width =) (rWidth)) to [Results v]
add (join (The area is) (rArea)) to [Results v]
add (join (The perimeter is) (rPerimeter)) to [Results v]
```

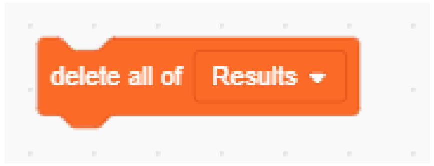
- To add a list, you click on the button *Make a List* in the Variable class. In this example, the name of the list is Results.



```
Calculator: Results
(empty)
+          length 0          =
```

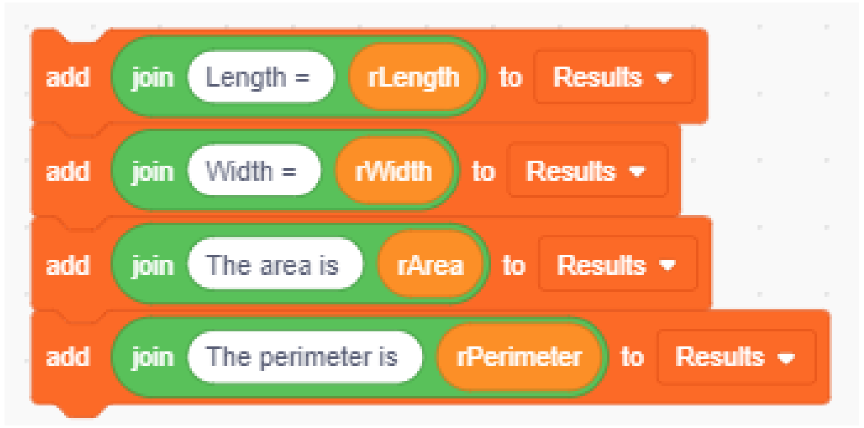
- An empty list will appear on the screen. The name of the sprite, and the name of the list will be displayed. The name Results was chosen, now it will be more clear to a user what the meaning of the output is.

- In other programming languages a List is called an *array*.



```
delete all of [Results v]
```

The [delete all of] block removes all lines in the List. Therefore when the program is executed a second time, the previous output will be deleted.



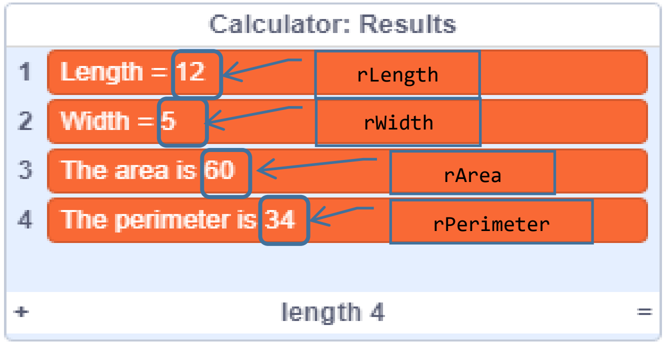
```
add join Length = rLength to Results
add join Width = rWidth to Results
add join The area is rArea to Results
add join The perimeter is rPerimeter to Results
```

The [add .. to] block adds the string created by the [join] block to the bottom of the List.

- The sprite was renamed to *Calculator.*
- Each [add] block added the ouput string to the bottom of the List.
- Each line in the List is numbered (in this example 1 to 4).
- The number of lines in the List is displayed at the bottom of the list (in this example it is 4).

Sometimes you will receive an example of the output of a program. You can use this output to help you to plan your program. Highlight all the values in the output where variables will have to be used.

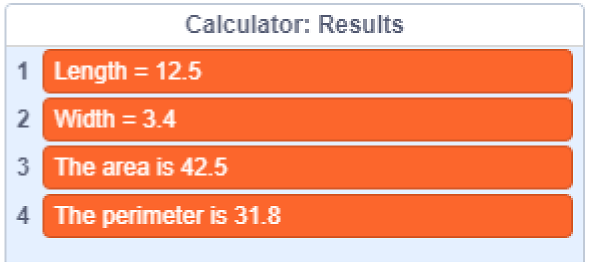
In the screenshot alongside, you can see which values are obtained from variables.



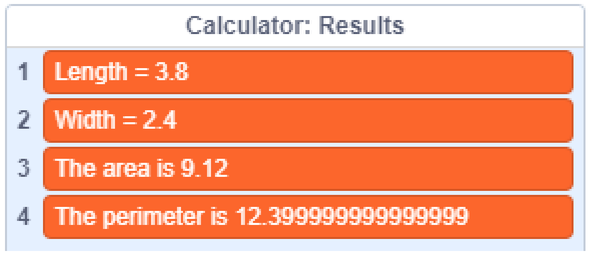
```
Calculator: Results
1 Length = 12
2 Width = 5
3 The area is 60
4 The perimeter is 34
+ length 4 =
```

Activity 25: Area and perimeter - Display output to a List

Create the program for the previous worked example. Use the values in the examples of output as test data.



```
Calculator: Results
1  Length = 12.5
2  Width = 3.4
3  The area is 42.5
4  The perimeter is 31.8
```



```
Calculator: Results
1  Length = 3.8
2  Width = 2.4
3  The area is 9.12
4  The perimeter is 12.399999999999999
```

Real values (also called floating point values), are not stored with complete accuracy. For example, the vaue 0.5 is stored as 0.5000000000000001. Therefore, when real values are displayed, the programming language approximate the values. However, the developers of Scratch did not build this functionality into Scratch when displaying values to a List. When you program in Java, you will learn how to prevent this type of display issue.

- Floating point values

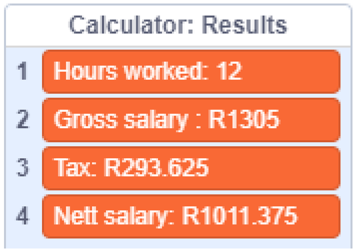
Activity 26: Wages – display output to a List

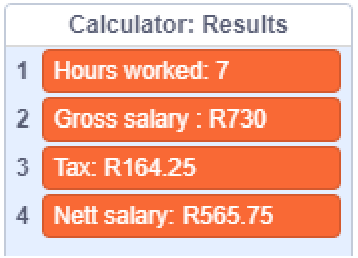
William is a temporary worker and receives a wage of R100 per hour for the first 5 hours worked but for any additional hours the hourly wage is increased by 15%. He has to pay tax at a rate of 22.5% of his gross income. The hours worked by him must be entered (e.g. 6.8). Assume he will work more than 5 hours. Calculate his gross income, tax to be paid, and netto income. The following is an example of the output you need to display. Use the numbers in the example output as test data.

Remember to plan

- Draw a diagram to represent the way the salary will be calculated.
- Indicate the variables in the examples of output.
- Determine the output and input variables, and use any number (e.g. 12) as input.
- Complete the calculation yourself. Ensure you are able to do the calculations correctly by comparing your answer to the test data provided. Write a complete algorithm, then write the Scratch code.

Example output.

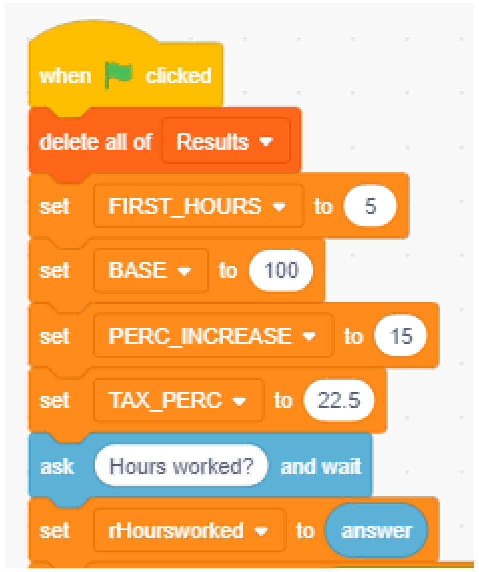

```
Calculator: Results
1  Hours worked: 12
2  Gross salary : R1305
3  Tax: R293.625
4  Nett salary: R1011.375
```


```
Calculator: Results
1  Hours worked: 7
2  Gross salary : R730
3  Tax: R164.25
4  Nett salary: R565.75
```

Certain values are ‘fixed’ – in this case:

- the rate for the first 5 hours (base rate),
- the percentage the hourly rate should be increased by, and
- the percentage tax.

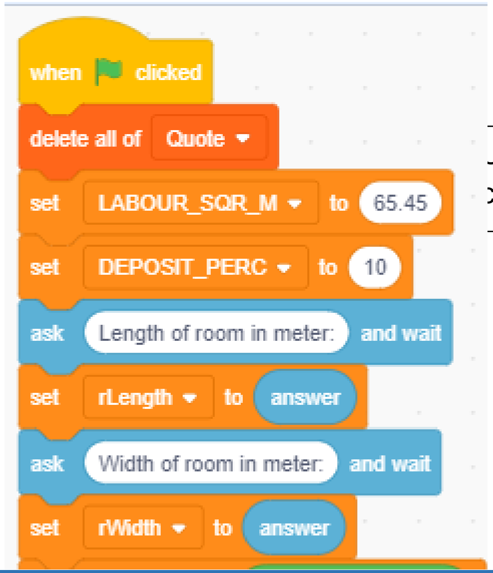
You can set these values at the beginning of your program, and use the variables in all calculations. In that way, if you decide to make changes to the values, you do not need to find all the ocurrences of the values in your code, you simply change the values allocated to the variables.


```
when [flag] clicked
delete all of Results
set FIRST_HOURS to 5
set BASE to 100
set PERC_INCREASE to 15
set TAX_PERC to 22.5
ask Hours worked? and wait
set rHoursworked to answer
```

In other programming languages, you indicate fixed values by using special key words, and the programming language will not allow you to make changes to these values. The values are then called *constants*.

Activity 27: Contractor fees – display output to a List

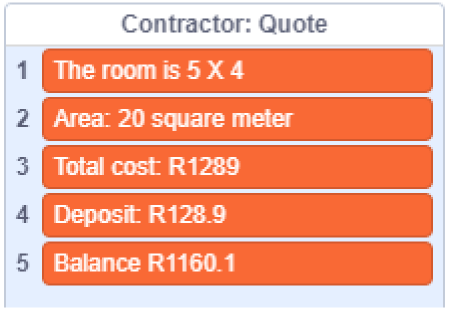
A contractor lays floor tiles at a tariff of R65.45 per square meter (m^2). Calculate the total cost to tile a room of a certain length and width in meter (as specified by the user). The contractor requires a deposit of 10% of the cost prior to commencing, and the balance on completion of the work. Calculate the deposit as well as the balance. Display the area, cost, deposit and the balance. Use fixed variables where relevant.



```
when [flag] clicked
delete all of Quote
set LABOUR_SQR_M to 65.45
set DEPOSIT_PERC to 10
ask Length of room in meter: and wait
set rLength to answer
ask Width of room in meter: and wait
set rWidth to answer
```

Use fixed values for the labour cost and percentage deposit.

Example of output.



```
Contractor: Quote
1 The room is 5 X 4
2 Area: 20 square meter
3 Total cost: R1289
4 Deposit: R128.9
5 Balance R1160.1
```

Circle each value in the ouput. Write the variable name you are going to use in your program next to each value.

Activity 28: Happy Doggie kennels

The Happy Doggie Kennels charges the following kennel fees:

For the first day	R60.00
For day 2 and more	R52.50 per day

The kennel also charges 25% of the total kennel fee to exercise the dogs each day. The owner should pay separately for food for the dog charged at a fee per day. Calculate and display the kennel and exercise fees, as well as the cost for the food. (All dogs will stay for at least two days.)

```
when the user clicks the green flag
       // Declare fixed values (constants)
```

- Kennel

```
       FIRSTDAY = 60
       OTHERDAYS = 52.5
       EXERCISEPERC = 25
       // Input
       display "How many days did the dog stay? "
       enter iNumDays
       display "What is the cost of food per day? "
       enter rCostFoodDay
       //Calculations
       rKennelFee =
       rExerciseFee  =
       rKennel_Exercise =
       rTotalFood =
       // Output to a List
       addToList "Kennel and exercise fee R", rKennel_Exercise
       addToList "Cost for food R", rTotalFood
end
```

Note

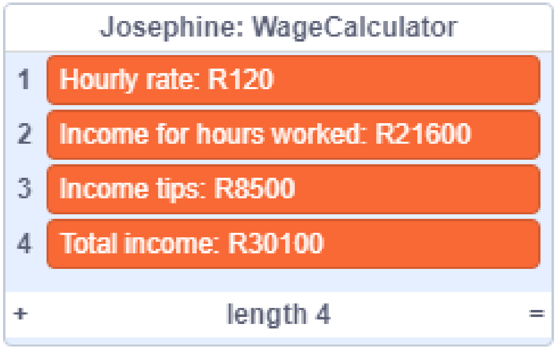
There are 4 variables used in the calculations (processing). Two of them are output variables (rTotalFood and rKennel_Exercise).

The other two variables are used in the calculations to determine the output values, but they are not displayed (rKennelFee and rExerciseFee). These type of variales are called *intermediate* variables.

Activity 29: Wage for 4 weeks

Josephine is working as a waitress at a restaurant:

- Read her hourly wage from the keyboard.
- She works 8 hours per day.
- She works 5 days a week.
- She also works every second Sunday for 5 hours.
- On Sundays she is paid double the hourly rate.
- She usually gets tips of approximately R50 per hour.



```
Josephine: WageCalculator
1  Hourly rate: R120
2  Income for hours worked: R21600
3  Income tips: R8500
4  Total income: R30100
+          length 4          =
```

Calculate her (approximate) income in a month that has 4 weeks.

An example of output is provided. Use the output to determine which values to calculate and display. Write a Scratch program based on the pseudocode.

Activity 30: Sell magwinya

Pulani made bakes magwinya to sell to fellow students. The muffins she does not sell, she gives away to students in need (so she does not receive money for them). She wants a program to calculate her profit per day.

The following values are constants:

- The cost price to bake one dozen.
- The price of one magwinya.

The following values should be entered for a specific day:

- The number of dozens of magwinya she baked.
- The number of magwinya she sold.

```
when the user clicks the green flag
      // Declare fixed values
      COSTDOZEN = 32
      COSTONE = 3.5
      // Input
      display "How many dozens of magwinya did she bake? "
```

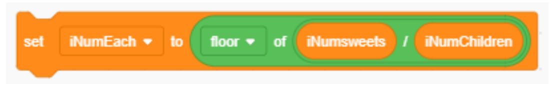
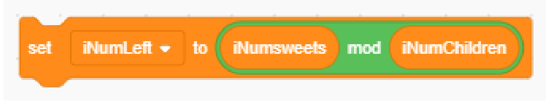
Calculate and display:

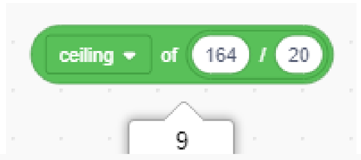
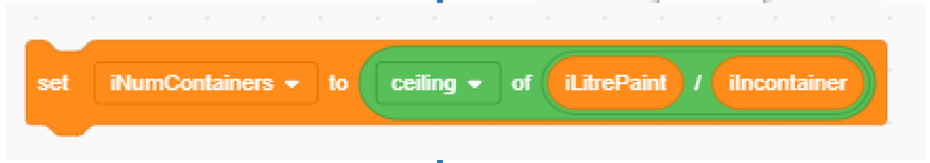
- how much money she needed to bake the magwinya,
- what percentage of magwinya did she sell,
- how much money she received from her sales, and
- how much profit she made.

```
      enter iNumDozens
      display "How many magwinya did she sell? "
      enter iNumMagwinyaSold
      // Processing
      rMoneyNeeded =
      rPercSold =
      rMoneyFromSales =
      rProfit =
      // Output
      addToList "Money needed was R" + moneyNeeded
      addToList "Percentage sold: " + percSold, "%"
      addToList "Money earned from sales R" + moneyFromSales
      addToList "Profit made R" + profit
end
```

Calculations with integer values

In Computational Matehematics you learned that many calculations require you to do calculations with integer numbers. The following are examples of various calculations where the results are integer values. The table contains the mathematical equation, as well as the pseudocode and Scratch program blocks necessary to do the calculations.

Mathematical scenario	Pseudocode	Scatch program block
We have 13 sweets, and need to divide it equally between 4 children. How many sweets wil each child receive? We need to use *integer division*. 13 \ 4 = 3 (3 * 4 = 12, so you can give each child 3 sweets) We can also use the *floor* notation to indicate that a value should be 'rounded down', or we can say the fraction part of the number should be removed (truncated). $\left\lfloor \frac{13}{4} \right\rfloor$	How will we calculate the number of sweets each child should receive for any number of sweets, and any number of children? Use the following variables: `iNumsweets; iNumChildren; iNumEach` $iNumEach = iNumSweets \mid iNumChildren$ or $iNumEach = \lfloor iNumSweets/iNumChildren \rfloor$	 set iNumEach to floor of iNumsweets / iNumChildren
We have 13 sweets, and need to divide it equally between 4 children. How many sweets will be left over after each child received the same number of sweets? We need to use the *modulus operator (MOD).*	How will we calculate the number of sweets that will remain for any number of sweets, and any number of children? Use the following variables: `iNumsweets; iNumChildren; iNumLeft`	 set iNumLeft to iNumsweets mod iNumChildren

13 MOD 4 = 1 (13 = 4 * 3 + 1, so you can give each child 3 sweets, and 1 sweet will remain).	$\text{iNumLeft} = \text{iNumSweets MOD iNumChildren}$ or $\text{iNumLeft} = \text{iNumSweets \% iNumChildren}$	
Joseph wants to paint his mother's house and calculated that he needs 164 litres of paint. However, the paint is only sold in 20 litre containers. How many containers of paint must he buy? 164/20 = 8.2 – therefore we need to buy 9 containers. The value 8.2 must be 'rounded up' to 9. In Mathematics we can use the *ceiling* notation to indicate that a value should be rounded up. $\left\lceil \frac{164}{20} \right\rceil$	How will we calculate the number of containers for any number of litres paint, and any size container. Use the following variables: *iLitrePaint ; iInContainer; iNumContainers* $\text{iNumcontainers} = \lceil \text{iLitrePaint}/\text{iInContainer} \rceil$	Programming languages have methods available to 'round a value up'. In Scratch we have the *ceiling* method.  ceiling of 164 / 20 9  set iNumContainers to ceiling of iLitrePaint / iIncontainer

Determine results for integer division and MOD

You can use any of the following methods to determine the results for the MOD and integer division (\) operators:

Example 1

$15 \div 6 = \boxed{2}$ remainder $\boxed{3}$

$15 = 6 * \boxed{2} + \boxed{3}$

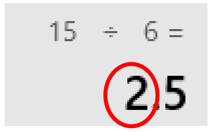
$15 \backslash 6 = \boxed{2}$

(The integer part of the division)

15 MOD 6 = $\boxed{3}$

(The remainder after long division)

On your pocket calculator the result of the division will be 2.5. The result of integer division is the number before the decimal point. Just ignore the fraction.


15 ÷ 6 =
2.5

Example 2

$5 \div 7 = \boxed{0}$ remainder $\boxed{5}$

$5 = 7 * \boxed{0} + \boxed{5}$

$5 \backslash 7 = \boxed{0}$

5 MOD 7 = $\boxed{5}$

Activity 31: Integer division and mod

Find the missing numbers:

35 = 6 * __ + _____	35 ÷ 6 = ____ remainder ______	35 \6 =	35 MOD 6 =
9 = 2 * ___+ _____	9 ÷ 2 = _________ remainder ______	9 \ 2 =	9 MOD 2 =
2 = 9 * ____+ ______	2 ÷ 9 = ______ remainder ______	2 \ 9 =	2 MOD 9 =
134 = 7 * ______ + ______	134 ÷7 = ______ remainder ______	134 \ 7	134 MOD 7 =
36 = 6 * ______ + ______	36 ÷ 6 = ______ remainder ______	36 \ 6 =	36 MOD 6 =
12 = 2.5 * ______ + ______	12 ÷ 2.5 = ______ remainder ______	12 \ 2.5 =	12 MOD 2.5 =

Activity 32: Chairs in student centre

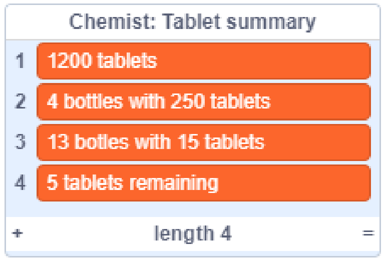
TUT has a certain number of chairs available to place in the student centre for a concert. Write an algorithm to calculate the maximum number of rows of 54 chairs each that can be filled with chairs. Calculate how many chairs will be left. (The number of available chairs must be read from the keyboard.)

Write a Scratch program based on the algorithm.

Activity 33: Tablets in bottles

A medicine factory has a certain number of tablets (read the number of tablets from the keyboard). They have two types of bottles. The one can hold 250 tablets, and the other 15 tablets. They want to fill all the bottles that can contain 250 tablets first, and then place the remaining tablets in the bottles that can hold 15 tablets. Calculate and display how many bottles of each size can be packed, and how many tablets will remain. Use the following variable names (you may have other variables as well if you need to):

iNumTablets; iNumB250; iNumB15; iRemTablets

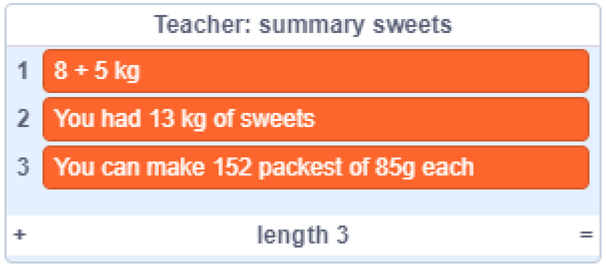


```
Chemist: Tablet summary
1  1200 tablets
2  4 bottles with 250 tablets
3  13 botles with 15 tablets
4  5 tablets remaining
+        length 4        =
```

Activity 34: Number of 85g packets

A school receives two containers of sweets, each containing a certain kilogram of sweets (read the mass of each from the keyboard). How many small packets of sweets of 85g each can be packed using the sweets from both containers?

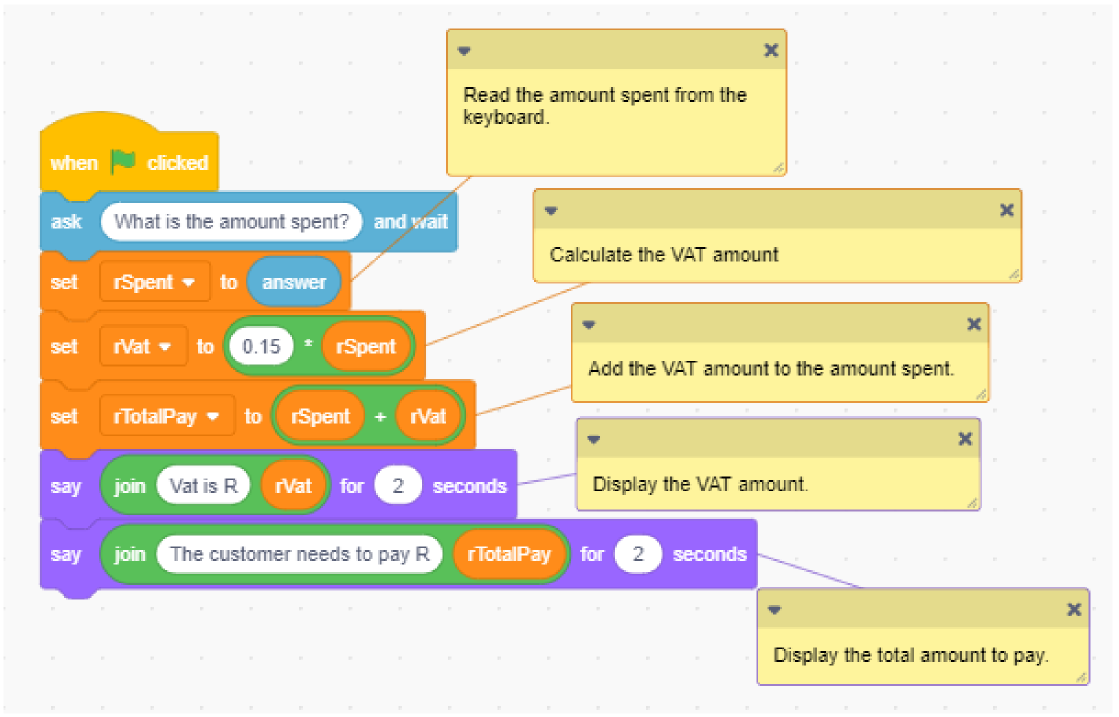
Write an algorithm to solve the problem, then write a Scratch program based on the algorithm.



```
Teacher: summary sweets
1  8 + 5 kg
2  You had 13 kg of sweets
3  You can make 152 packest of 85g each
+        length 3        =
```

Obtain more input after output

In Activity 13: , we wrote the following program:



```
when [flag] clicked
ask (What is the amount spent?) and wait
set rSpent to (answer)
set rVat to ((0.15) * (rSpent))
set rTotalPay to ((rSpent) + (rVat))
say (join (Vat is R) (rVat)) for (2) seconds
say (join (The customer needs to pay R) (rTotalPay)) for (2) seconds
```

Read the amount spent from the keyboard.

Calculate the VAT amount

Add the VAT amount to the amount spent.

Display the VAT amount.

Display the total amount to pay.

When you buy a number of articles, you wait to see the amount you need to pay on the cash register, before you hand over money to the cashier. For example, if you have to pay R45.00, you can pay with a R50.00 note, and receive R5.00 change. Or you can pay using a R100.00 note, and receive R55.00 change.

We are going to adjust this program:

- this time we are going to ask the user what amount the customer pays after displaying the amount,
- then calculate and display the amount of change to be returned to the customer.

Write an algorithm for the program.

```
when the user clicks the green flag
      // Constant value
      VAT_PERC = 0.15
      // Input
      ask "What is the amount spent? "
```

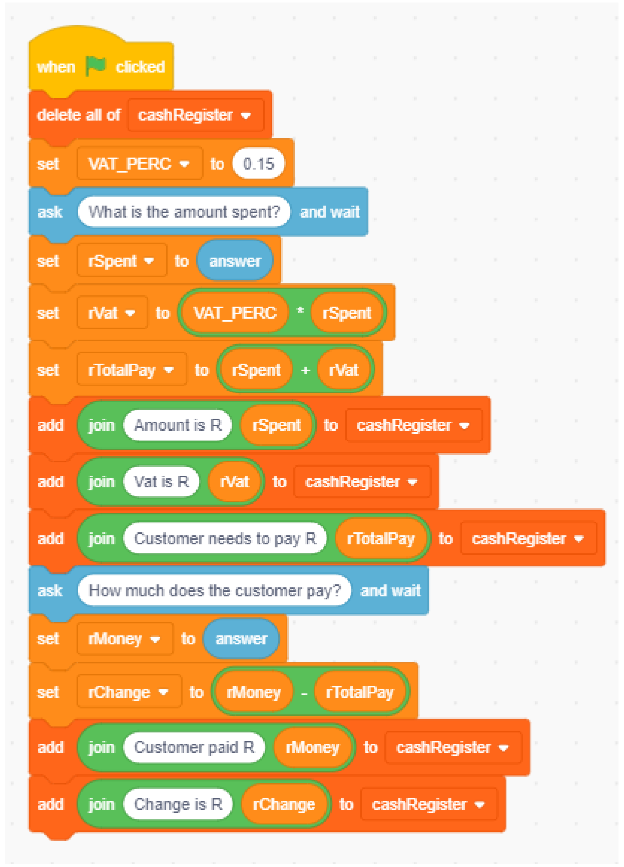
```
        enter rSpent
        // Processing
        rVat = VAT_PERC * rSpent
        rTotalPay = rSpent + rVat
        // Output to a List
        addToList "Amount is R" + rSpent
        addToList "VAT R" + rVat
        addToList "Customer needs to pay R" + rTotalPay
        // Input – user needs to see the total amount before deciding what to pay
        ask "How much does the customer pay? "
        enter rMoney
        // Processing
        rChange = rMoney – rTotalPay
        // Output to a List
        addToList "Customer paid R" + rMoney
        addToList "Change is R" + rChange
end
```

The amount to be paid will be displayed in the List. The user of the program can see what needs to be paid.

The program will display a prompt to ask the user for the amoount the customer will pay, and read a value from the keyboard (input).

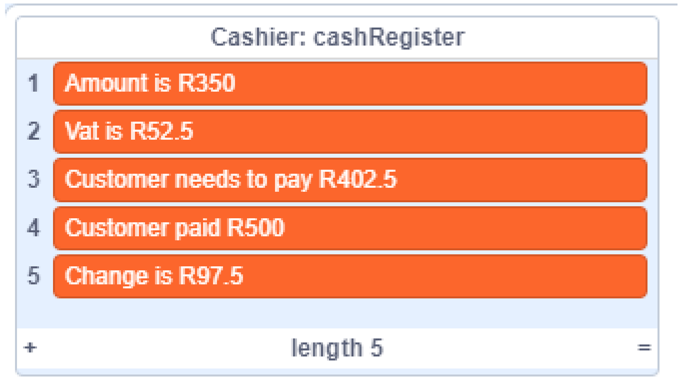
Translate the algorithm to programming code.

- The complete program will look as follows:



```
when [flag] clicked
delete all of cashRegister
set VAT_PERC to 0.15
ask What is the amount spent? and wait
set rSpent to answer
set rVat to VAT_PERC * rSpent
set rTotalPay to rSpent + rVat
add join Amount is R rSpent to cashRegister
add join Vat is R rVat to cashRegister
add join Customer needs to pay R rTotalPay to cashRegister
ask How much does the customer pay? and wait
set rMoney to answer
set rChange to rMoney - rTotalPay
add join Customer paid R rMoney to cashRegister
add join Change is R rChange to cashRegister
```

Example of output.



```
Cashier: cashRegister
1 Amount is R350
2 Vat is R52.5
3 Customer needs to pay R402.5
4 Customer paid R500
5 Change is R97.5
+        length 5        =
```

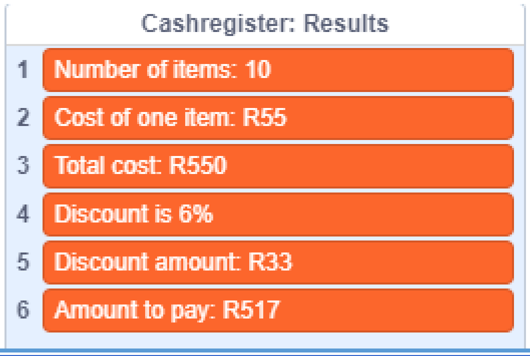
- The sprite was renamed *Cashier.*
- Each [add] block added the ouput string to the bottom of the List.
- Each line in the List is numbered (in this example 1 to 5).
- The number of lines in the List is displayed at the bottom of the list (in this example it is 5).
- Scratch does not show 2 values after the decimal point as we would prefer when displaying a monetary value. For example, display R52.50 – it displays R52.5. In other programming languages it is possible to format the output with 2 decimal values quite easily.

Activity 35: Calculate cost and discount

Write pseudocode for a program that will do the following:

- Read the number of items a customer buys and the price of one item. (All items have the same price.)
- Display the cost of all the items, and ask the user what percentage discount must be given.
- Display the discount amoount, as well as the final amount to be paid.

An example of output is provided. Use the output to determine which values to calculate and display.
Write a Scratch program based on the pseudocode.


Cashregister: Results

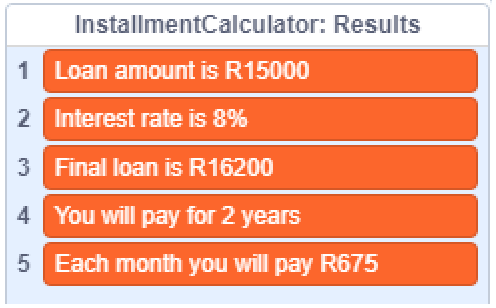
```
1 Number of items: 10
2 Cost of one item: R55
3 Total cost: R550
4 Discount is 6%
5 Discount amount: R33
6 Amount to pay: R517
```

Activity 36: Calculate installment per month

Write pseudocode for a program that will do the following:

- Read the amount a person wants to loan from a bank.
- Read the interest rate that will be charged.
- Display the total loan amount.
- Ask the user over how many years the person wants to pay back the loan.
- Calculate and display what the installment for each month should be.

An example of output is provided. Use the output to determine which values to calculate and display.
Write a Scratch program based on the pseudocode.


InstallmentCalculator: Results

```
1 Loan amount is R15000
2 Interest rate is 8%
3 Final loan is R16200
4 You will pay for 2 years
5 Each month you will pay R675
```

Activity 37: Knowledge questions

a) What is a variable?
b) What is the difference between a radiobutton and a checkbox?
c) How can a *readout* be used to help you to find errors in your program?

d) How can you determine whether a variable has stage scope or sprite scope?
e) What is an IPO table?
f) What is an algorithm?
g) Which character(s) are used to indicate a comment in al algorithm?
h) Why is it useful to declare some values as fixed values (constants) in a program?
i) How will you explain the difference between integer division and MOD to another student?
j) Would you recommend that a programmer adds comments to a program? Motivate your answer.

Activity 38: Skills checklist

Ensure you can do the following:

	I know how ✓
a) Define input and output for a given programming problem.	
b) Complete an IPO table as part of designing an algorithm.	
c) Choose valid variable names based on best practices standards.	
d) Identify fixed-value variables (constants) and write it in the correct section of an algorithm.	
e) Suggest valid data types for variables by indicating the data type in an algorithm.	
f) Apply the rules of the order of processing when writing pseudo code for a mathematical calculation.	
g) Design an algorithm to solve a basic mathematical problem requiring sequential steps, present it in pseudo code, and implement it in Scratch providing correct output.	
h) Add comments in a program to clarify the flow of a program.	
i) Display the value of a variable on the stage using readouts.	

j) Decide whether a variable should have sprite or stage scope.	
k) Design programming solutions to solve basic mathematical problems using mathematical operators such as addition, subtraction, multiplication, division, integer division and MOD.	
l) Apply the rules of the order of processing by using stacking of code blocks in Scratch.	
m) Display multiple lines of output on a list on the stage.	
n) Write programming solutions where more data must be obtained after output has been displayed by the program.	

Reflection

1. What was the purpose of the lesson content you re discussing?
2. What did you learn that will advance your future learning?
3. What new knowledge and skills did you learn today? (Tip: To help you to answer this question – if someone asks you tonight – “So, what did you learn in class today?” – what will you answer them? Your answers can, for example, start with “I am now able to….I managed to … I learned how to ….)
4. What did you figure out on your own?
5. What is still challenging or confusing for you to get your mind around?
6. What did you find most challenging today?
7. Are you now confident that you have mastered that difficult concept / skill?
8. If not – what are you going to do to make it work?
9. How was your thinking extended or pushed today?
10. How are the knowledge and skills you learned today connected to what you already knew?
11. Did you discover a connection today that you were not aware of previously?
12. What did you enjoy doing and why?
13. What do you think could be done differently in today’s lesson?
14. What did you discover that may make someone else’s life easier in future classes?

(Sackson, 2011)

(Harvard Graduate School of Education, n.d.)

Sources consulted

2018 Scratch Foundation. (2015, June 28). Retrieved from Code-to-Learn Foundation: http://scratchfoundation.org/about-us

Coetzee, C., & Wassermann, U. (2016). Reflections on the application of scaffolding principles to support transfer of mathematical skills to entry-level algorithm design in a novice programming course. *EdMedia.* Vancouver.

edSurge. (2015, June 28). Retrieved from Learn to Code, Code to Learn: https://www.edsurge.com/n/2013-05-08-learn-to-code-code-to-learn

Erasmus, H. G., & Pretorius, C. M. (2012). *Basic Programming Principles* (2nd ed.). Cape Town, South Africa: Pearson Education South Africa (Pty) Ltd.

Farrel, J. (2002). *Programming Logic and Design, Introductory* (2nd ed.). Boston: Course Technology.

Liang, D. Y. (2013). *Introduction to Java Programming: Comprehensive Version* (9th ed.). Harlow, England: Pearson Education Limited.

Murach, J. (2017). *Murach's Java programming* (5th ed.). United States of America: Mike Murach & Associates, Inc.

Sackson, E. (2011, June 11). *10 ways to encourage student refelction.* Retrieved December 2, 2019, from What Ed said: https://whatedsaid.wordpress.com/2011/06/11/10-ways-to-encourage-student-reflection-2/

ScratchEd. (2015, June 28). *What is computational thinking?* Retrieved from ScratchEd: http://scratched.gse.harvard.edu/ct/defining.html

Tshwane University of Technology. (2019). FPIDS01:Practical Study Guide. (U. Wassermann, Compiler) Pretoria, Gauteng: Tshwane University of Technology.

Wassermann, U., Gentle, E., Gibson, K., Noomé, C., Van Zyl, S., & Zeeman, M. (2014). *IT is gr8! @ Grade 11: Delphi.* Pretoria, South Africa: Study Opportunities.

Wassermann, U., Noomé, C., Gentle, E., Gibson, K., Macmillan, P., & Zeeman, M. (2011). *IT is gr8! @ Grade 10.* Pretoria, South Africa: Study Opportunities.