# Introduction to Programming Glossary

*By Reuben Wilson ( 9988289 )*

## Programming Overview

Firstly, consideration for the definition of a variable is required.

Variable: A type of artefact, used for the purpose of storing a single value in. The value can be varied!

Variables are used in procedures/functions locally and, based on the definition above; their value can be varied. Consider my DrawBike program, where we are passing the procedure DrawBike some parameters to begin with; they're defined in the comments.

```
// Call DrawBike passing it the parameters ColorRed, 50, 10
DrawBike(ColorRed, 50, 10);
// Call DrawBike passing it the parameters ColorBlue, 75, 60
DrawBike(ColorBlue, 75, 60);
```

By declaring these two sets of unique parameters, an understanding is established that the DrawBike procedure must then behave in two different ways. Firstly, take a look at the DrawBike procedure below:

```
// Create a new procedure called 'DrawBike'
// This procedure is going to take three parameters
// clr is the colour we will pass the procedure to draw the bike in
// x and y and the positioning points for the bike, this data will be passed to the procedure
procedure DrawBike(clr: Color; x, y: Integer);
var
// Declare variables for the position of the bikes wheels as Integers
wheel1X, wheel1Y, wheel2X, wheel2Y: Integer;
// Declare variables for the positioning of the frame as Integers
frameX1, frameY1, frameX2, frameY2, frameX3, frameY3: Integer;
// Declare variables for the positioning of the handle bars as 2D points on a cartesian plane
barX, barY: Point2D;

begin
    // This section deals with drawing the first wheel (back wheel)
    // Calculate the values for the variables, then assign those vales to their respective variable
    wheel1X := (x + radius);
    wheel1Y := y + (bikeHeight - radius);
    // Draw a circle with the passed colour at the points calculated and assigned to the variables wheel1X and wheel1Y
    // radius defines, you guessed it - the radius for the circle being drawn
    DrawCircle(clr, wheel1X, wheel1Y, radius);

    // This section deals with drawing the second wheel (front wheel)
    // Calculate the values for the variables, then assign those vales to their respective variable
    wheel2X := x + (bikeWidth - radius);
    wheel2Y := wheel1Y;
    // Draw a circle with the passed colour at the points calculated and assigned to the variables wheel2X and wheel2Y
    // radius defines, you guessed it - the radius for the circle being drawn
    DrawCircle(clr, wheel2X, wheel2Y, radius);

    // This section deals with drawing the bike frame
    // Calculate the values for the variables, then assign those vales to their respective variable
    frameX1 := wheel1X;
    frameY1 := wheel1Y;
    frameX2 := x + Round(bikeWidth / 2);
    frameY2 := y + Round(radius / 2);
    frameX3 := wheel2X;
    frameY3 := wheel2Y;
    // Draw a triangle with the passed colour at the points calculated and assigned to the variables
    // frameX1, frameY1, frameX2, frameY2, frameX3, frameY3
    DrawTriangle(clr, frameX1, frameY1, frameX2, frameY2, frameX3, frameY3);

    // This section deals with drawing the bike handle bars
    // Calculate the values for the variables, then assign those vales to their respective variable
    barX := PointAt(wheel2X, y);
    barY := PointAt(wheel2X, wheel2Y);
    // Draw a line with the passed colour from the points calculated and assigned to the variables barX, barY
    DrawLine(clr, barX, barY);
end;
```

As you can see by the comments above, the values of the variables declared within the procedure DrawBike are varied by the input of parameters. The values are directly calculated and assigned from given data! This is a good example of how the data in a variable can vary within a procedure.

Declaring and using variables locally (within a procedure or function) is much safer than using variables, which have been declared globally (variables which belong to the entire program).

Why? Because variables, which are declared locally, can only be accessed and written to by the entity to which they belong. Variables declared in the program can be accessed from anywhere within the program and <u>varied</u> by <u>any</u> procedure or function that uses it.

## Programming Terms:

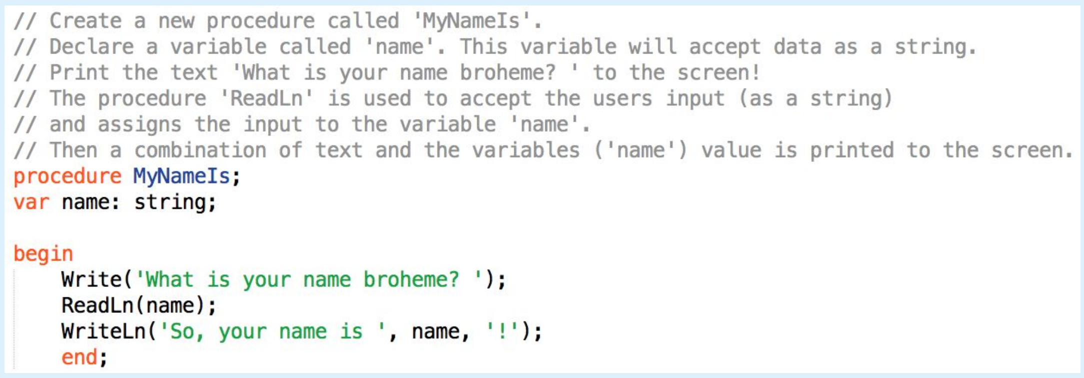
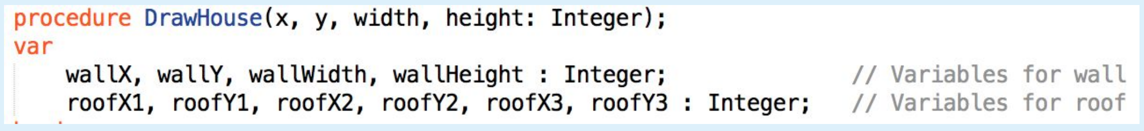
The following terms have specific meanings for software developers. It is important to understand these terms, as they will be used to communicate ideas.

| Term | Description |
|---|---|
| **Parameter** | A parameter is used to pass information, or data to a procedure. |
| **Local Variable** | These are variables, which exist within functions and procedures, meaning, that variable belongs to the entity in which it is coded. |
| **Global Variable** | These are variables which belong to the entire program, meaning, these variables can be read and written to from anywhere within the program. |

## Program

| | **Details / Answer** |
|---|---|
| **A program contains…** | All sorts of goodies including fun stuff like procedures, functions, variables and parameters. If you refer to the section ‘Procedures can contain’ just below, you’ll find an information log provided on how functions, parameters and variables are handy. |

## Procedure

| | **Details / Answer** |
|---|---|
| **Procedures can contain** | A plethora of instructions, ranging from printing a string on the screen as text or asking for data input, such as your name, loading images, sounds etc.<br><br>Variables are often used within procedures as storage devices for data. Variables are handy because you can <u>vary</u> the data they store. E.g. I want my program to ask me for my name and then store that string in a container <u>or</u> variable. This snippet of code below outlines exactly how you would go about storing a piece of data in a variable.<br><br><br>`// Create a new procedure called 'MyNameIs'.`<br>`// Declare a variable called 'name'. This variable will accept data as a string.`<br>`// Print the text 'What is your name broheme? ' to the screen!`<br>`// The procedure 'ReadLn' is used to accept the users input (as a string)`<br>`// and assigns the input to the variable 'name'.`<br>`// Then a combination of text and the variables ('name') value is printed to the screen.`<br>`procedure MyNameIs;`<br>`var name: string;`<br><br>`begin`<br>`    Write('What is your name broheme? ');`<br>`    ReadLn(name);`<br>`    WriteLn('So, your name is ', name, '!');`<br>`    end;`<br><br>Using parameters in conjunction with your procedures is also useful because by doing so, you can pass data to your procedure. E.g. In the snippet of code below a procedure called ‘DrawHouse’ is being declared and it is going to be passed data, or, parameter values for ‘x’, ‘y’, ‘width’ and ‘height’.<br><br><br>`procedure DrawHouse(x, y, width, height: Integer);`<br>`var`<br>`    wallX, wallY, wallWidth, wallHeight : Integer;           // Variables for wall`<br>`    roofX1, roofY1, roofX2, roofY2, roofX3, roofY3 : Integer;   // Variables for roof` |
| **Procedures are similar to…** | Procedures and functions are similar in the sense that they can both be declared for use with parameters and variables. The differentiating factor between the two is that; functions return a <u>single</u> value, meaning they are useful for defining any mathematics that a program may need. |

## Procedure Execution:

When a procedure is called from within a program, the computer then reads that procedures coded instructions and executes them.

These instructions may include variables and parameters. Refer to the snippet of code below to see how a procedure is being passed parameters:

```
// Call DrawBike passing it the parameters ColorRed, 50, 10
DrawBike(ColorRed, 50, 10);
```

Now that you can visualise the parameters being passed to the procedure, you can understand from the code below how a procedure (in this case DrawBike) executes in response. The code also outlines how this procedure incorporates variables and their use in execution.

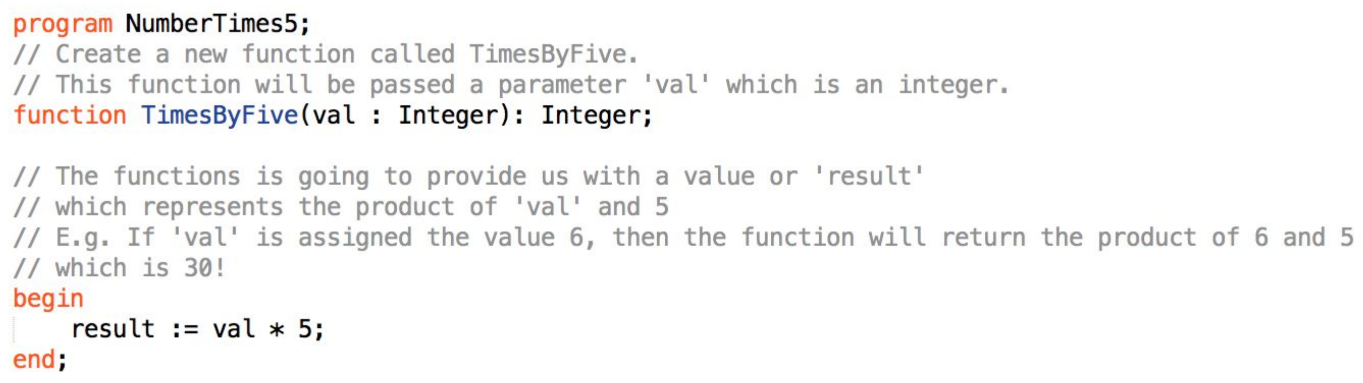
```
procedure DrawBike(clr: Color; x, y: Integer);
var
// Declare variables for the position of the bikes wheels as Integers
wheel1X, wheel1Y, wheel2X, wheel2Y: Integer;
// Declare variables for the positioning of the frame as Integers
frameX1, frameY1, frameX2, frameY2, frameX3, frameY3: Integer;
// Declare variables for the positioning of the handle bars as 2D points on a cartesian plane
barX, barY: Point2D;

begin
    // This section deals with drawing the first wheel (back wheel)
    // Calculate the values for the variables, then assign those vales to their respective variable
    wheel1X := (x + radius);
    wheel1Y := y + (bikeHeight - radius);
    // Draw a circle with the passed colour at the points calculated and assigned to the variables wheel1X and wheel1Y
    // radius defines, you guessed it - the radius for the circle being drawn
    DrawCircle(clr, wheel1X, wheel1Y, radius);

    // This section deals with drawing the second wheel (front wheel)
    // Calculate the values for the variables, then assign those vales to their respective variable
    wheel2X := x + (bikeWidth - radius);
    wheel2Y := wheel1Y;
    // Draw a circle with the passed colour at the points calculated and assigned to the variables wheel2X and wheel2Y
    // radius defines, you guessed it - the radius for the circle being drawn
    DrawCircle(clr, wheel2X, wheel2Y, radius);

    // This section deals with drawing the bike frame
    // Calculate the values for the variables, then assign those vales to their respective variable
    frameX1 := wheel1X;
    frameY1 := wheel1Y;
    frameX2 := x + Round(bikeWidth / 2);
    frameY2 := y + Round(radius / 2);
    frameX3 := wheel2X;
    frameY3 := wheel2Y;
    // Draw a triangle with the passed colour at the points calculated and assigned to the variables
    // frameX1, frameY1, frameX2, frameY2, frameX3, frameY3
    DrawTriangle(clr, frameX1, frameY1, frameX2, frameY2, frameX3, frameY3);

    // This section deals with drawing the bike handle bars
    // Calculate the values for the variables, then assign those vales to their respective variable
    barX := PointAt(wheel2X, y);
    barY := PointAt(wheel2X, wheel2Y);
    // Draw a line with the passed colour from the points calculated and assigned to the variables barX, barY
    DrawLine(clr, barX, barY);
end;
```

## Function

| | **Details / Answer** |
|---|---|
| **A function is …** | A coded instruction, which returns a specific value. |
| **Functions can contain** | Functions can contain problem-solving solutions, like mathematical equations. |
| **Functions are similar to…** | Are similar to procedures in the respect that you can pass them parameters and declare variables within them. |
| **Only difference is…** | Functions are designed to return a <u>single</u> value. |

### Example 1: NumberTimes5

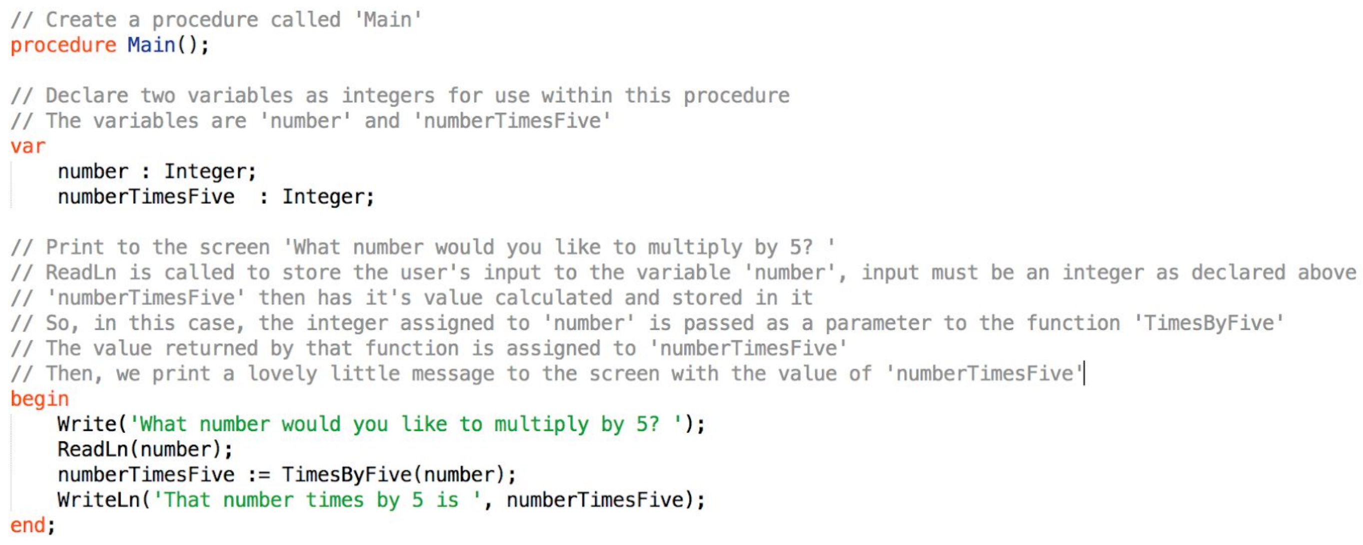
Refer to this function example and the code comments for an understanding of what it does.



```
program NumberTimes5;
// Create a new function called TimesByFive.
// This function will be passed a parameter 'val' which is an integer.
function TimesByFive(val : Integer): Integer;

// The functions is going to provide us with a value or 'result'
// which represents the product of 'val' and 5
// E.g. If 'val' is assigned the value 6, then the function will return the product of 6 and 5
// which is 30!
begin
    result := val * 5;
end;
```

### Function Execution:

Refer to the snippet of code below, which shows the function NumberTimes5 being called.



```
// Create a procedure called 'Main'
procedure Main();

// Declare two variables as integers for use within this procedure
// The variables are 'number' and 'numberTimesFive'
var
    number : Integer;
    numberTimesFive  : Integer;

// Print to the screen 'What number would you like to multiply by 5? '
// ReadLn is called to store the user's input to the variable 'number', input must be an integer as declared above
// 'numberTimesFive' then has it's value calculated and stored in it
// So, in this case, the integer assigned to 'number' is passed as a parameter to the function 'TimesByFive'
// The value returned by that function is assigned to 'numberTimesFive'
// Then, we print a lovely little message to the screen with the value of 'numberTimesFive'
begin
    Write('What number would you like to multiply by 5? ');
    ReadLn(number);
    numberTimesFive := TimesByFive(number);
    WriteLn('That number times by 5 is ', numberTimesFive);
end;
```

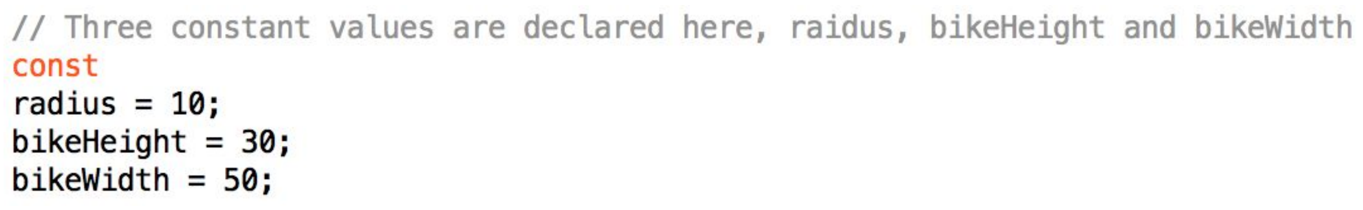
When this function is called from within a program, the computer then reads the coded instructions for the function and executes them. The comments in the procedure (Main) and function (TimesBy5) explain exactly what happens when that function is called.

## Constant

| | Details / Answer |
|---|---|
| **A constant is …** | A fixed value in a program that does not change. |
| **Constants are similar to…** | They're similar to variables in the sense that they're containers for data. |
| **The main difference is…** | Data in a variable can be varied, data in a constant cannot. |

### Example 1: radius, bikeHeight, bikeWidth

In this snippet of code, I've clearly demonstrated how you define constants.



```
// Three constant values are declared here, raidus, bikeHeight and bikeWidth
const
radius = 10;
bikeHeight = 30;
bikeWidth = 50;
```

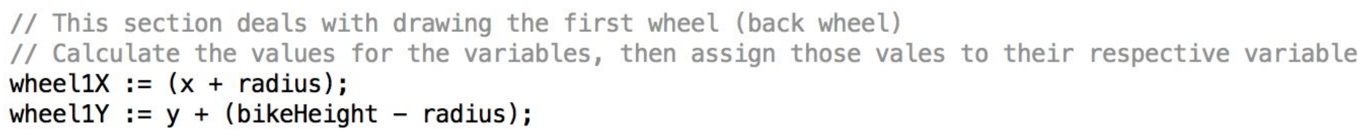
These constants are used in my DrawBike program. The code for said program is attached, feel free to peruse (I'm sure you will). The comments outline how the functions are used.

## Variable

| | Details / Answer |
|---|---|
| **A variable is …** | An artefact we use to assign calculated data to. Think of them as containers for storage. |
| **Variables in a program are called:** | Global variables, meaning they can be accessed from anywhere within the program. |
| **Variables in a procedure or function are called:** | Local variables, meaning they can only be accessed from within the function/procedure they belong to. |
| **Variables passed to procedures and functions are called:** | Parameters! A handy implementation of passing data directly to a procedure/function. |

### Example 1: wheel1X, wheel1Y

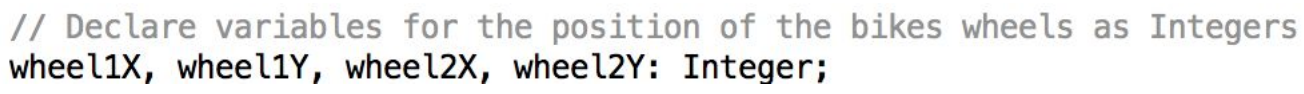
This is an example of two variables used in my DrawBike program. For sake of understanding, DrawBike is coded to draw two wheels, a frame and some handlebars. So, that in mind, these two variables are responsible for the position (x,y) of the back wheel of my bike!



```
// This section deals with drawing the first wheel (back wheel)
// Calculate the values for the variables, then assign those vales to their respective variable
wheel1X := (x + radius);
wheel1Y := y + (bikeHeight - radius);
```

**Local Variable Creation:**

The snippet of code below demonstrates the declaration of variables; these variables will accept a single value as an integer.



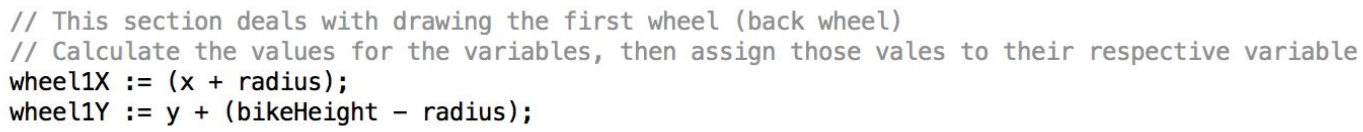
```
// Declare variables for the position of the bikes wheels as Integers
wheel1X, wheel1Y, wheel2X, wheel2Y: Integer;
```

When a computer creates a value for a variable, it follows these steps:

1. Calculate the value
2. Assign the calculated value to the variable

So, if you refer to the small section below previously used above, you can understand for wheel1X that the computer:

1. Calculates the value of (x + radius)
2. Assigns the calculated value to the variable wheel1X.

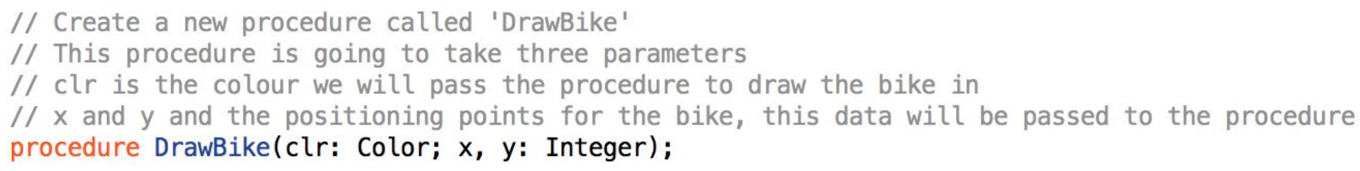


```
// This section deals with drawing the first wheel (back wheel)
// Calculate the values for the variables, then assign those vales to their respective variable
wheel1X := (x + radius);
wheel1Y := y + (bikeHeight - radius);
```

**Parameter Creation:**

The snippet of code below demonstrates the declaration of parameters within a procedure; they are:

| | |
|---|---|
| clr | – This parameter will take a colour |
| x & y | - These parameters will take integer values only |



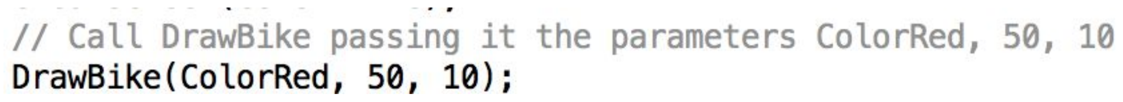
```
// Create a new procedure called 'DrawBike'
// This procedure is going to take three parameters
// clr is the colour we will pass the procedure to draw the bike in
// x and y and the positioning points for the bike, this data will be passed to the procedure
procedure DrawBike(clr: Color; x, y: Integer);
```

So, when a computer creates a value for a parameter, it follows these steps:

1. Calculate the value
2. Assign the calculated value to the parameter.

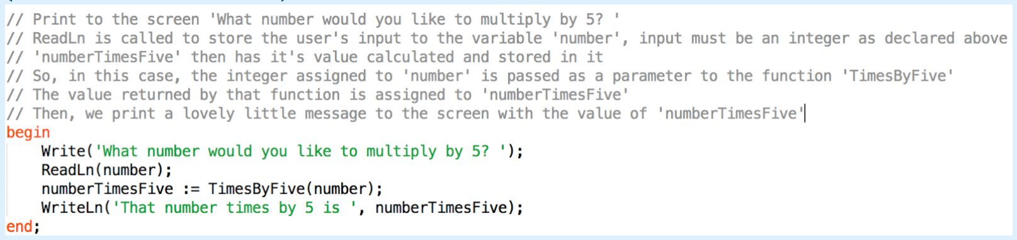
So, if you refer to the small section below, you can see the procedure DrawBike being passed parameters, which correlate with their data types. The computer:

1. Calculates the value ColorRed and assigns it to the parameter clr
2. Calculates the values 50, 10 and assigns them to them parameters x and y!



```
// Call DrawBike passing it the parameters ColorRed, 50, 10
DrawBike(ColorRed, 50, 10);
```

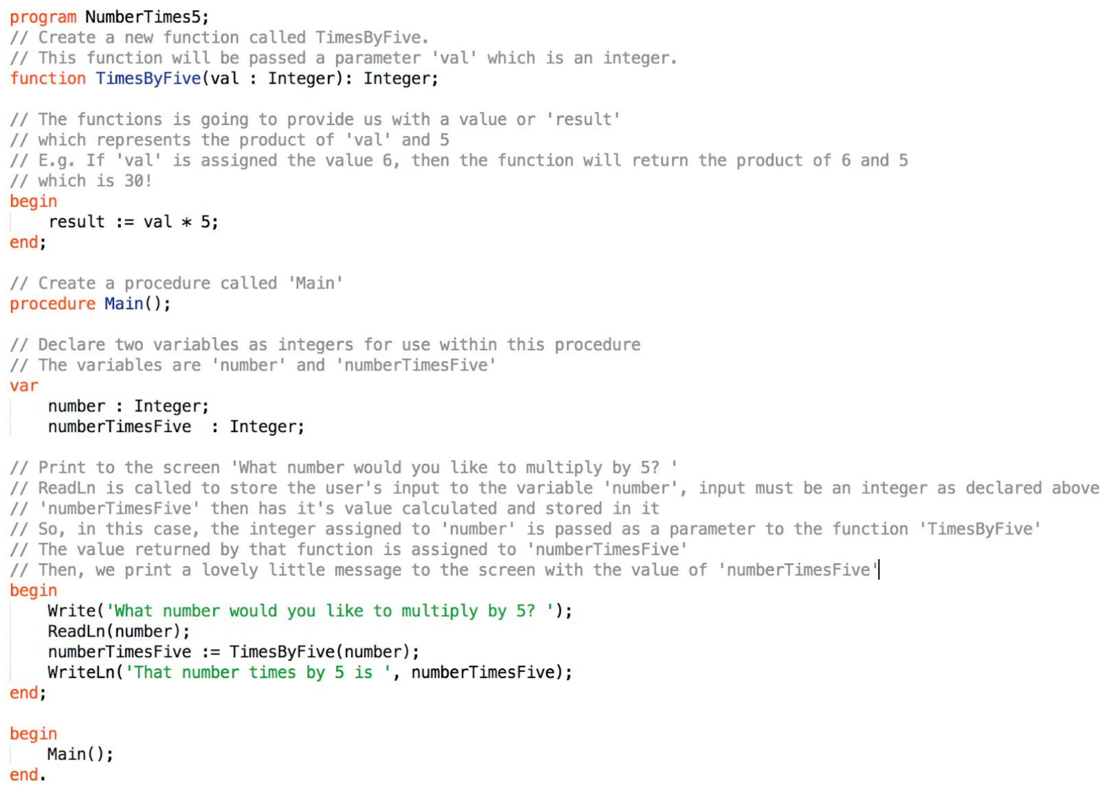
## Function Call

| | Details / Answer |
|---|---|
| **A function call …** | Is when the computer reaches an instruction to execute a function within a program. Below is an example of a function (TimesBy5) being called to calculate a value and assign it to a variable (numberTimesFive):  |
| **Is actually an expression as …** | Assignment statement. |
| **Parameters allow you to …** | Pass data to a procedure/function to determine how a procedure will execute. |
| **The value returned …** | The value returned by a procedure/function being passed parameters depends on how that procedure/function deals with the data being passed. |

```
// Print to the screen 'What number would you like to multiply by 5? '
// ReadLn is called to store the user's input to the variable 'number', input must be an integer as declared above
// 'numberTimesFive' then has it's value calculated and stored in it
// So, in this case, the integer assigned to 'number' is passed as a parameter to the function 'TimesByFive'
// The value returned by that function is assigned to 'numberTimesFive'
// Then, we print a lovely little message to the screen with the value of 'numberTimesFive'
begin
    Write('What number would you like to multiply by 5? ');
    ReadLn(number);
    numberTimesFive := TimesByFive(number);
    WriteLn('That number times by 5 is ', numberTimesFive);
end;
```

### Example 1: Assignment Statement with Function Call

The value returned from this function (TimesByFive) call is stored in the variable (numberTimesFive) using an assignment statement.



```
program NumberTimes5;
// Create a new function called TimesByFive.
// This function will be passed a parameter 'val' which is an integer.
function TimesByFive(val : Integer): Integer;

// The functions is going to provide us with a value or 'result'
// which represents the product of 'val' and 5
// E.g. If 'val' is assigned the value 6, then the function will return the product of 6 and 5
// which is 30!
begin
    result := val * 5;
end;

// Create a procedure called 'Main'
procedure Main();

// Declare two variables as integers for use within this procedure
// The variables are 'number' and 'numberTimesFive'
var
    number : Integer;
    numberTimesFive  : Integer;

// Print to the screen 'What number would you like to multiply by 5? '
// ReadLn is called to store the user's input to the variable 'number', input must be an integer as declared above
// 'numberTimesFive' then has it's value calculated and stored in it
// So, in this case, the integer assigned to 'number' is passed as a parameter to the function 'TimesByFive'
// The value returned by that function is assigned to 'numberTimesFive'
// Then, we print a lovely little message to the screen with the value of 'numberTimesFive'
begin
    Write('What number would you like to multiply by 5? ');
    ReadLn(number);
    numberTimesFive := TimesByFive(number);
    WriteLn('That number times by 5 is ', numberTimesFive);
end;

begin
    Main();
end.
```

## Actions Performed

When a function call is executed the computer performs the following steps:

1. The computer reaches the procedure call.
2. The computer places the function in the stack.
3. The computer then goes back through the code to locate the functions instructions.
4. Once the computer knows the instructions for that function, it executes them.

## Assignment Statement

| | **Details / Answer** |
|---|---|
| **An assignment statement …** | Is used to assign calculated data to a variable. |
| **On the left side of the assignment you put a …** | The variable you wish to have the data assigned to. |
| **On the right side of the assignment you put …** | The data that you wish to assign to the variable. |

### Example 1: Assigning data to wheel2X

This example assignment statement assigns the calculated value x + (bikeWidth – radius) to the variable wheel2X.

```
wheel2X := x + (bikeWidth - radius);
```

### Example 2: Assigning data to frameX2

This example assignment statement assigns the calculated value  x + Round(bikeWidth / 2) to the variable frameX2.

```
frameX2 := x + Round(bikeWidth / 2);
```

### Example 3: Assigning data to airSpeedVelocity

This example assignment statement assigns the calculated value of a function AirSpeed and assigns it to the variable airSpeedVelocity.

```
airSpeedVelocity := AirSpeed(inputFrequency, calcAmplitude);
```

Here is how the function Airspeed is instructed:

```
// Create a new function called AirSpeed
// This function will take 2 parameters, frequency and amplitude as Integers
// These integers are then converted to the data type Double
function AirSpeed(frequency, amplitude: Integer): Double;

// The functions result is the product of frequency and amplitude over strohaulNumber
// ***strohaulNumber is declared as a constant***
begin
    result := Round((frequency * amplitude) / strohaulNumber);
end;
```

## Actions Performed

When an assignment statement is executed the computer performs the following steps:

1. Calculates the value, which is to be assigned (value to the right).
2. Assigns the calculated value to a variable (value to the left).

## Core Exercise 2 – Expressions

1. What are the values and types of the following expressions? Types will be either Integer, Double, or String.

| *Question* | *Expression* | *Given that* | *Value* | *Type* |
|---|---|---|---|---|
| a | 5 | | | |
| b | a | a := 2.5 | | |
| c | 1 + 2 * 3 | | | |
| d | a + b | a := 1 and b := 2 | | |
| e | 2 * a | a := 3 | | |
| f | a * 2 + b | a := 1.5 and b := 2 | | |
| g | a + 2 * b | a := 1.5 and b := 2 | | |
| h | (a + b) * c | a := 1, b := 1 and c := 5 | | |
| i | 'Fred' + ' Smith' | | | |
| j | a + ' Smith' | a := 'Wilma' | | |
| k | a + b * 2 | a := 1.0 and b := 2 | | |

*Table 1: Sample expressions to evaluate*

2. What data type is most appropriate to store the following?

| *Question* | *Desired Data* | *Type* *(Integer, Double, or String)* |
|---|---|---|
| a | A person's name | |
| b | Number of students in a class | |
| c | Average age of a group of people | |
| d | A temperature in Celsius | |
| e | The name of a subject | |
| f | Runs scored in a cricket match | |
| g | A student's ID number | |
| h | A person's phone number | |
| i | The cost of an item | |

## Core Exercise 3 – Hand Execution Assignment

```
var                          // starts the variable declarations
    a: Integer;              //
begin                        //
    a := 2;                  //
    WriteLn(a - 1, ' ', a);  //
end.
```

---

```
var                          // starts the variable declarations
    a, b, c: Integer;        //
begin                        //
    a := 10;                 //
    b := 20;                 //
    c := a + b;              //
    WriteLn('c is ', c);     //
end.
```

---

```
var                          // starts the variable declarations
    a, b, c, d, e: Integer;  //
begin                        //
    a := 10;                 //
    b := 20;                 //
    c := 30;                 //
    d := a;                  //
    e := 50;                 //

    b := a;                  //
    a := a + b;              //
    c := e - c;              //
    d := a;                  //
    e := e - 1;              //

    WriteLn(a, ' ', b, ' ', c, ' ', d, ' ', e); //
end.
```

## Core Exercise 4 – Hand Execution Parameter passing

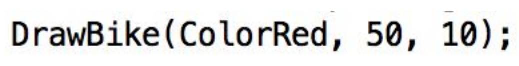
```
function DoubleVal(val: Integer) : Integer; //
begin                           //
    result := val + val;        //
end;

var                             // starts the variable declarations
    a, b: Integer;              //
begin                           //
    a := 2;                     //
    b := DoubleVal(a);          //
    WriteLn( DoubleVal(b), b, a);       //
end.
```

---

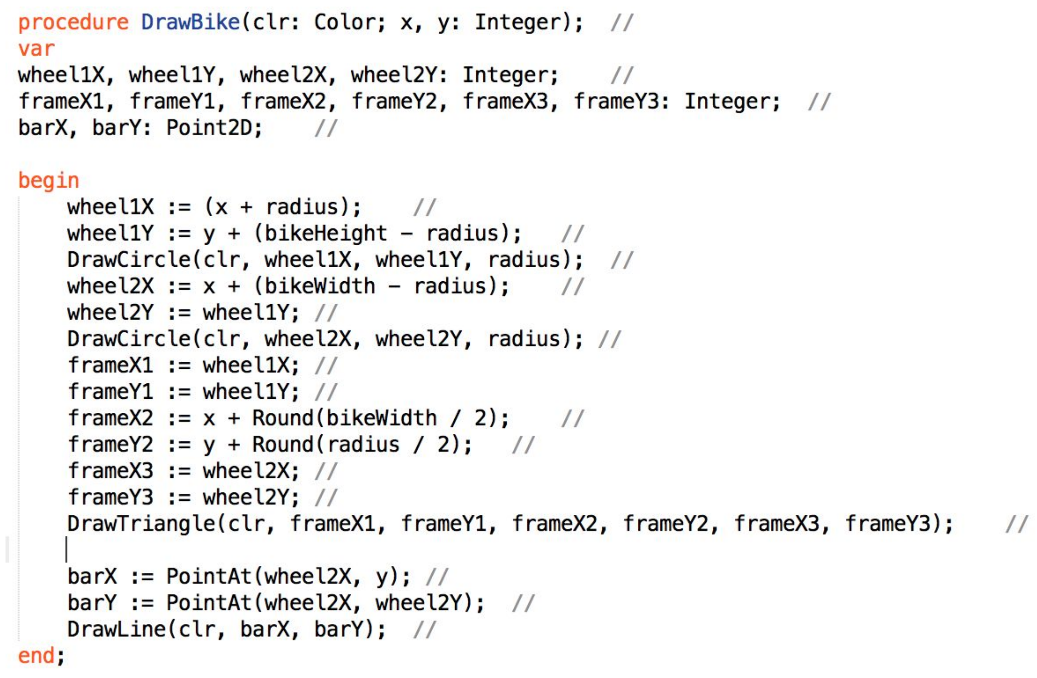
```
procedure DoubleIt(var val: Integer); //
begin                           //
    val := val + val;           //
end;

var                             // starts the variable declarations
    a: Integer;                 //
begin                           //
    a := 2;                     //
    DoubleIt(a);                //
    WriteLn( a );               //
end.
```

## Core Exercise 5 – DrawBike

Hand execute with:

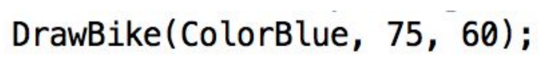


```
DrawBike(ColorRed, 50, 10);
```

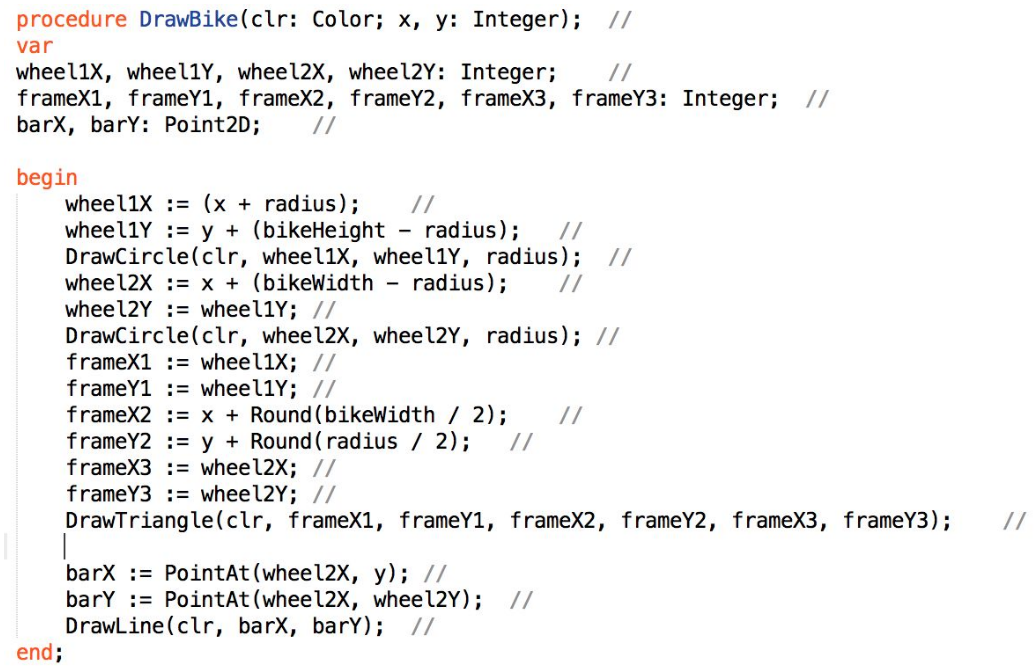


```
procedure DrawBike(clr: Color; x, y: Integer);  //
var
wheel1X, wheel1Y, wheel2X, wheel2Y: Integer;    //
frameX1, frameY1, frameX2, frameY2, frameX3, frameY3: Integer;  //
barX, barY: Point2D;    //

begin
    wheel1X := (x + radius);    //
    wheel1Y := y + (bikeHeight - radius);   //
    DrawCircle(clr, wheel1X, wheel1Y, radius);  //
    wheel2X := x + (bikeWidth - radius);    //
    wheel2Y := wheel1Y; //
    DrawCircle(clr, wheel2X, wheel2Y, radius); //
    frameX1 := wheel1X; //
    frameY1 := wheel1Y; //
    frameX2 := x + Round(bikeWidth / 2);    //
    frameY2 := y + Round(radius / 2);   //
    frameX3 := wheel2X; //
    frameY3 := wheel2Y; //
    DrawTriangle(clr, frameX1, frameY1, frameX2, frameY2, frameX3, frameY3);    //

    barX := PointAt(wheel2X, y); //
    barY := PointAt(wheel2X, wheel2Y);  //
    DrawLine(clr, barX, barY);  //
end;
```

Hand execute with:

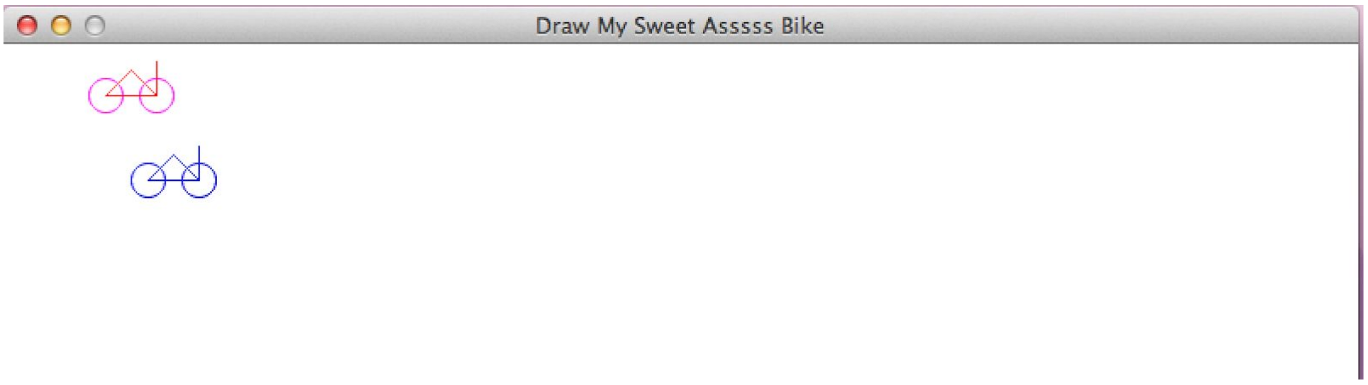


```
DrawBike(ColorBlue, 75, 60);
```



```
procedure DrawBike(clr: Color; x, y: Integer);  //
var
wheel1X, wheel1Y, wheel2X, wheel2Y: Integer;    //
frameX1, frameY1, frameX2, frameY2, frameX3, frameY3: Integer;  //
barX, barY: Point2D;    //

begin
    wheel1X := (x + radius);    //
    wheel1Y := y + (bikeHeight - radius);   //
    DrawCircle(clr, wheel1X, wheel1Y, radius);  //
    wheel2X := x + (bikeWidth - radius);    //
    wheel2Y := wheel1Y; //
    DrawCircle(clr, wheel2X, wheel2Y, radius); //
    frameX1 := wheel1X; //
    frameY1 := wheel1Y; //
    frameX2 := x + Round(bikeWidth / 2);    //
    frameY2 := y + Round(radius / 2);   //
    frameX3 := wheel2X; //
    frameY3 := wheel2Y; //
    DrawTriangle(clr, frameX1, frameY1, frameX2, frameY2, frameX3, frameY3);    //

    barX := PointAt(wheel2X, y); //
    barY := PointAt(wheel2X, wheel2Y);  //
    DrawLine(clr, barX, barY);  //
end;
```

A screenshot of my BikeDrawing.pas running



Please find my code for this program attached at the back.

## Extension Exercise 1 – AirSpeedVelocity.pas

Please find the code for this program attached at the back.

## Extension Exercise 2 – Custom Program: Data