Introduction to the Turing Programming Language

To this point in the course, we have used application programs. Application programs are created by professional programmers and used by users like us to solve problems. Examples include Word Perfect, Microsoft Office, FileMaker Pro, virus checkers and all game programs.

Programming languages are used by programmers to create application programs. These application programs can be simple like the ones we will create in this course or very complex like the ones listed in the previous paragraph. Examples of programming languages include Turing (which we will use), Java, C++, Basic and Pascal.

When you start Turing by selecting it from the Start menu, you are presented with the Editor Window in which you will type your program. Your cursor (the small triangle) is located at the top left of the Editor Window. Typing in Turing is similar to typing in a word processing application program like Word or Apple Works. Many of the keys like Enter, Backspace, Insert, Delete, and the navigation keys (End, Home, Page Up, Page Down, and the arrow keys) have the same function in Turing as they do in a word processor.

The programs you create are made up of a series of program commands. A program command is an instruction which tells the computer how to work with data. After you type your program into the Editor Window, you have the computer execute it (run it) by selecting the Run command button at the top of the Editor Window or by pressing the F1 key. If all the commands are correct, the computer will follow your instructions and show the results in a separate Run Window. If there is an error, the computer will highlight the problems in your program and give you a message describing what the problem is.

Exercise #1

The following easy program will allow the user to input their first name into the computer and have it outputted on the monitor. Start Turing and type the text that is not italicized in the Editor Window. The bolded text is the actual Turing command. The plain text is the data. The italicized text describes what the command does. Be careful to input it exactly as it appears. When you finish inputting the program, run it using by selecting the Run command button at the top of the Editor Window or by pressing the F1 key!

```turing
var name : string % A storage spot for the user’s first name.
put “Enter your first name” % Outputs a message on the screen for the user.
get name % Stops the computer and allows the user to input their first name.
put “Your name is “, name
```

If you finish the program, improve it by having the user input their last name, age and average and have this data outputted. When you finish the program, save it in a new folder called Programming as Names. A program is saved using the File/Save As.. command.

Turing Graphics

An easy way to get started with Turing is to draw graphics. When drawing graphics in Turing, the Run Window is divided into pixels which can be thought of as tiny dots of lights on the monitor. Each pixel has an x-coordinate and y-coordinate which tells the computer where to draw the pixel on the screen. Each pixel must also have a colour. The pixel at the bottom left of the window has the coordinate (0,0).
In the diagram above, the pixel has the x-coordinate of 100 and the y-coordinate of 150. The number of pixels on the window depends on the size of the Run Window used.

**Turing Drawing Commands**

Listed below are some of the drawing commands in Turing. Note that the letters in the brackets represent data that you must provide as shown in the examples.

`drawdot (x, y, c)` - Draws a coloured dot at the pixel
  - x is the x-coordinate
  - y is the y-coordinate
  - c is the colour.
  - For example, `drawdot (50, 20, red)` will draw a red dot at the pixel with the x-coordinate of 50 and the y-coordinate of 20. (Note you will have to look very closely!)

`drawbox (x1, y1, x2, y2, c)` - Draws a coloured box
  - x1 is the x-coordinate of the bottom left pixel of the box
  - y1 is the y-coordinate of the bottom left pixel of the box
  - x2 is the x-coordinate of the top right pixel of the box
  - y2 is the y-coordinate of the top right pixel of the box
  - c is the colour of the box
  - For example, `drawbox (250, 100, 300, 400, blue)` will draw a blue box. The bottom left corner will have the x-coordinate of 250 and the y-coordinate of 100. The top right corner will have the x-coordinate of 300 and the y-coordinate of 400.

`drawfillbox (x1, y1, x2, y2, c)` - Draws a coloured filled box
  - Data is the same as `drawbox`
drawoval (x, y, xRad, yRad, c) - Draws an oval
  - x is the x-coordinate of the pixel at the centre of the oval
  - y is the y-coordinate of the pixel at the centre of the oval
  - xRad is the length of the x radius of the oval (ie. the length of the radius of the oval along the x axis)
  - yRad is the length of the y radius of the oval (ie. the length of the radius of the oval along the y axis)
  - c is the colour of the oval.
  - For example, drawoval (150, 100, 100, 50, magenta) will draw a magenta oval. The centre of the oval will have the x-coordinate of 150 and the y-coordinate of 100. The x radius will be 100 pixels long while the y radius will be 50 pixels long.
  - Note that if the x radius and the y radius are the same, a circle will be drawn.

drawfilloval (x, y, xRad, yRad, c) - Draws a filled oval
  - Data is the same as drawoval

Exercise #2
Type the following small graphic drawing program. Save the program as Few Shapes

```
put "The largest value of x is: ", maxx % Outputs the largest possible x-coordinate
put "The largest value of y is: ", maxy % Outputs the largest possible y-coordinate
drawdot (50, 20, red)
drawbox (250, 100, 300, 350, blue)
drawfillbox (450, 150, 400, 350, yellow)
drawoval (150, 100, 100, 50, magenta)
drawfilloval (500, 100, 100, 50, green)
```

Improve your program by drawing:
1) A black dot at pixel 30, 40.
2) A purple box with the bottom left at pixel 40, 67 and the top right at pixel 89, 93. (Note that pixel 40, 67 means the pixel has an x-coordinate of 40 and a y-coordinate of 67).
3) A red filled box with the bottom left at pixel 350, 67 and the top right at pixel 503, 93.
4) A magenta oval with centre at pixel 150, 100, an x radius of 40 and a y radius of 50.
5) A brown oval with centre at pixel 550, 350, an x radius of 30 and a y radius of 30.
6) Any other shape you want. Try the drawline, drawmapleleaf and the drawstar commands. They all have the same data as drawbox.

Re-save your program as Few Shapes.

Exercise #3
Drawing shapes one at a time is time consuming. You will create a program which will automatically draw many shapes using a loop. In programming, a loop will allow the computer to automatically repeat some commands. This program also uses variables. A variable is a storage spot which is used to store data in a program. This data can be changed at any time. Type the program below and save it as Many Shapes.

```
var xCoord : int % A variable (storage spot) for the x coordinate
var yCoord : int % A variable (storage spot) for the y coordinate
var shapeColour : int % A variable which stores the colour
```
for count : 1 .. 300  % The start of a loop which will repeat the following commands 300 times.
    randint (xCoord, 1, 400)  % Will assign a random value between 1 and 400 in xCoord.
    randint (yCoord, 1, 400)  % Will assign a random value between 1 and 400 in yCoord.
    randint (shapeColour, 1, 255)  % Will assign a random value between 1 and 255 in shapeColour.
    drawdot (xCoord, yCoord, shapeColour)
    locate (1,1)
    put "Dot #", count  % Outputs the value of the counter variable
    delay (50)
end for  % Ends the loop

delay (2000)  % Delays the program for 2 seconds

for count : 1 .. maxcolour
    locatexy (275, 375)
    put "This is colour #: ", count
    drawfillbox (300, 300, 350, 330, count)
    delay (100)
end for

Exercise #4 - Car Race
You are now going to create a simple race game in a series of steps. In each step, add the appropriate commands and then run the program. Check off each step as you complete and re-save your program as Race.

___ Step #1
Draw a red filled oval with an x radius of 10 and a y radius of 10 at position 50, 200.

___ Step #2
Draw a white filled oval with an x radius of 10 and a y radius of 10 at position 50, 200. Note that nothing appears when you run your program. Why? ____________________________________________________________________________

___ Step #3
To solve the problem in Step #2, add the following command between the two drawfilloval commands: delay (100). You should see a flashing red dot. What is the purpose of the delay command? __________________________________________________________________________________________

___ Step #4
To make the program a little more interesting, make the flashing red dot repeat 20 times. Do this by adding a loop to the program as well as another delay command after you draw the white circle. Your program should look like the following:

    for count : 1 .. 20
        drawfilloval (50, 200, 10, 10, red)
        delay (100)
        drawfilloval (50, 200, 10, 10, white)
        delay (100)
    end for
How can you change the colour of the ball? _________________________________________
How can you make the program draw the flashing ball 50 times? ________________________
How can you increase the amount of time the red circle appears on the screen before it is erased?

Step #5
A ball which stays in the same place is rather boring. A moving ball would be much more interesting to look at. We can do this by creating a variable called xRed by adding the following command at the beginning of the program:

```
var xRed := 50  % Creates a variable to store integers with a starting value of 50
```

This will create a storage spot for data called xRed which will store the x position of the red ball. It will start the x coordinate at 50 but this will change.

Change your drawfilloval command which draws the red oval so that it looks as below:

```
drawfilloval (xRed, 200, 10, 10, red)
```

Run the program. Notice it runs exactly the same as it did before the change was made. Why?

Step #6
What we need to do is increase the value stored in the xRed. This is done using the following command after the white ball is drawn:

```
xRed += 3  % The value in the variable xRed is increased by 3 every time through the loop
```

Run the program. Notice the flashing ball moves. To eliminate the flashing and just have the red ball move, erase the 2\textsuperscript{nd} delay and run your program again.

Step #7
A race without a finish line is somewhat useless. Draw a finish line by adding a command at the beginning of your program to draw a black line with the starting x and y coordinates of 500, 0 and the ending x and y coordinates of 500, 480 (Hint: Use the drawline command). Run your program.

Notice that your ball does not reach the finish line. To fix this, change the \texttt{for} command so that it loops 300 times.

Step #8
A race program with only one competitor is not very useful. Draw a second, blue ball of the same size with the same starting x coordinate but a starting y coordinate of 300.
Hints: 1) You will need to create a 2nd variable much like you created a variable for your red ball. Call it xBlue

2) The command to draw the blue ball will be the same as the one to draw the red one except for the starting y position.

3) Remember, you will have to erase this ball after it is drawn much like you did with the red ball.

4) Remember as well you will have to increase the value of the xBlue variable every time through the loop.

---

Step #9

The problem with the program to this point is that both balls move at exactly the same speed every time through the loop - not much of a race. The speed of competitors must be random. To make the race more realistic, you will change your program so that the movement of both “racers” is random.

First a variable to store the amount the oval will move every time through the loop must be created. Use the following command at the beginning of the program:

```
var moveRed := 0  % Creates a variable which will store integer values starting at 0
```

You will also have to create a variable for the amount the blue ball will move.

Next, the program will need to assign a random value to these two move variables. This is done using the randint command. Input the following command at the beginning of your loop (ie. after the for command):

```
randint (moveRed, 1,6)  % Assigns a random value of between 1 and 6 to the moveRed variable
```

You will also have to have a randint command for the blue ball.

Finally, the positions of the two balls which are stored in the xRed and xBlue variables will have to be changed by the random amount. This is done for the red ball by changing the following command:

```
xRed += 3
```

to:

```
xRed += moveRed
```

Do the same for the blue ball.
To do this, the program will have to “make a decision”. This is done using the if statement. Add the following commands to your program at the end of the loop (just before the end for command):

```turing
if xRed > 500 then
  put "Red is the winner"  % Outputs the message only if xRed > 500
  exit  % Exits the loop
end if  % Finishes the if statement
```

Add a similar command for the blue ball. Re-save your program as Racing Ovals.

### Step #11

While the race program is coming along, having cars or horses (or some other things) racing would make it even more realistic. You can import JPEG or BMP graphics into Turing quite easily and make them move.

Copy two graphic files of cars called Car One.jpg and Car Two.jpg from the teacher assignment folder to your Programming folder.

Next, add the following commands at the top of your program before your loop:

```turing
var carOne := Pic.FileNew ("Car One.jpg")
var carTwo := Pic.FileNew ("Car Two.jpg")
```

Add the following commands within your loop.

```turing
Pic.Draw (carOne, xRed, 200, picCopy)
Pic.Draw (carTwo, xBlue, 300, picCopy)
```

Finally get rid of the commands you do not need. Which ones are these? __________________

Run your program and save it as Racing Cars. You may want to change your messages, etc. so they fit the program (ie. “Car #1 wins” instead of “Red Wins”)

### Step #12

Improve your program by:
- Adding a start line.
- Adding more and/or different items to race.
- Have the winning item “celebrate” - be creative.
- Have the items have to return to the start line in order to win.
- Add a timer to time the items.
- Any other improvement you can think of.