Writing Games using Python Turtle

GETTING STARTED

What will you do?

In this tutorial, you will learn how to write code in **Python** and create a 1982 Tron-like game using the **Python Turtle** tool. You can find more information about the history of the game here: https://en.wikipedia.org/wiki/Tron_(video_game).

The aim is to move a player around the game area using the direction keys on your keyboard. The game ends if the player either collides with the boundary of the game area or there is a path overlap.

What do you need?

You will need a computer with the Python programming language installed on it. You can either use a Raspberry Pi or a computer running Windows or Mac OS.

The easiest way to code in Python is to use an Integrated Development Environment (IDE). You can either use **IDLE** (the standard Python Integrated Development Environment) or **Thonny Python IDE**. Both IDLE and Thonny use **Python 3**.

If you are using a Raspberry Pi, you can either open **IDLE** or **Thonny Python IDE** by following these steps:

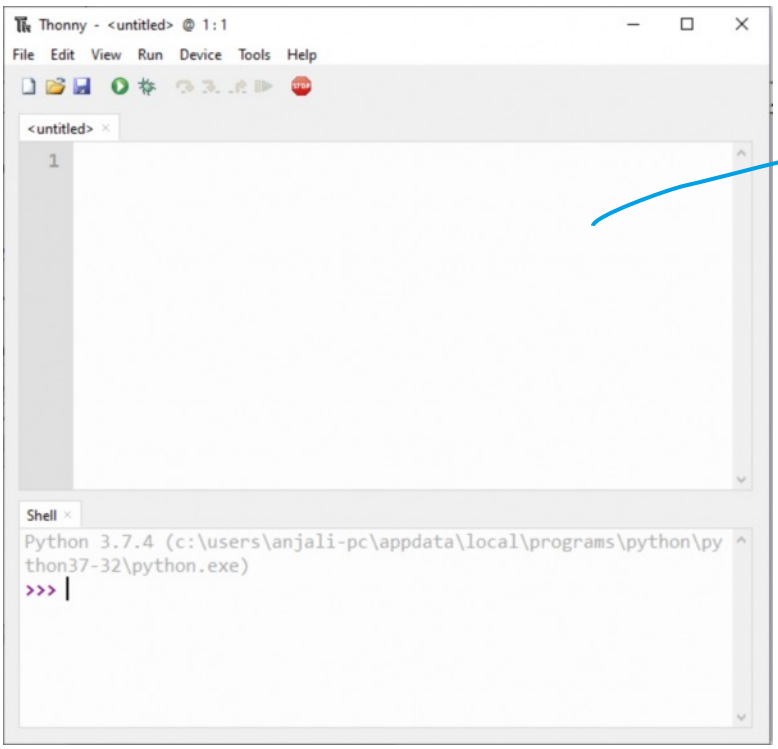
- Open the **applications menu** (**MENU** button) and select **Programming**;
- Click on **Python 3 (IDLE) or Thonny Python IDE**

Otherwise, if you are using a computer running Mac OS or Windows, you can download **IDLE** from https://www.python.org/downloads/ and install the latest version on your machine.

GETTING STARTED

IDE

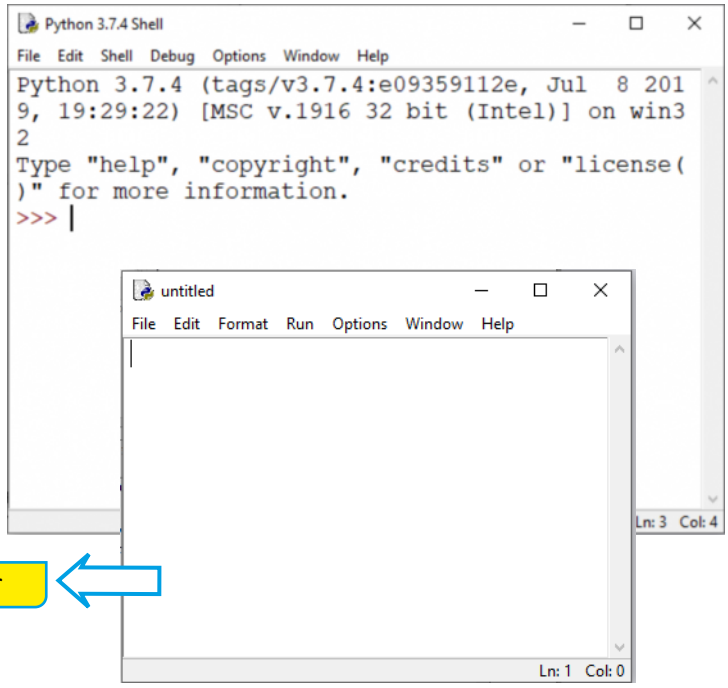
Depending on which IDE you are using, you will see one of the two interfaces shown below:



Start typing your code here

Thonny IDE

IDLE



Python Shell

Text Editor

Once the Python Shell is open in IDLE, click on **FILE** and then **NEW FILE**. This will open another window, the **TEXT EDITOR**, where you can start to write your Python program ...

FRED THE TURTLE

In the text editor window, **CLICK ON THE BLANK SPACE** to start typing.

Start by importing the Python turtle library in to your program. Type the following line:

```
import turtle
```

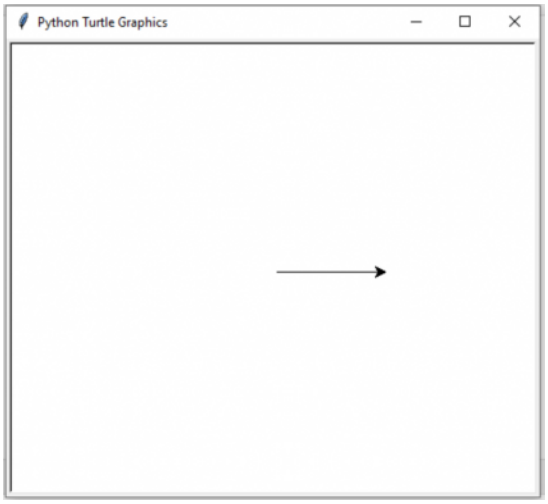
The player turtle can be given a name, we're going to call ours 'fred':

```
fred = turtle.Turtle()
```

Finally, make the turtle move around the screen. Let's start with a simple forward movement…

```
fred.forward(100)
```

This should make the 'turtle' move forward 100 pixels. To see if this works, you need to run the program. To do this, **PRESS F5** on the keyboard.



You should be asked to **SAVE** your program first. Click **OK** and **GIVE IT A NAME** (you can call it "game").

When the code runs, the Python Turtle Graphics windows should come into focus and display the turtle moving forward 100 'steps'!

Let us now play around with the background a bit. In true 1980s style, let us use a black background. Type in the new lines of code. Your final code should look like this:

```
import turtle

#Setup turtle screen
stage = turtle.Screen()
stage.bgcolor("black")
stage.title("Welcome to my game")

#Create player turtle
fred = turtle.Turtle()
fred.forward(100)
```

Creating a 'Screen' object to control the turtle window

Setting the turtle window background to black

Setting the title of the turtle window

Press F5 to run the program. We have a problem - you can't see Fred on a black background!

CREATING THE GAME AREA

We now change our ***player turtle***, ***Fred's*** characteristics.

Let's change it's colour first so that it becomes visible. Add this line of code after you have created the ***player turtle***.

We can also change the shape of the ***player turtle***. The 'turtle' that you get on the screen is the 'classic' turtle from Python's Turtle library. We can change that so that the ***player turtle*** looks like an actual turtle!

After adding the two new lines of code, your program should look like this:

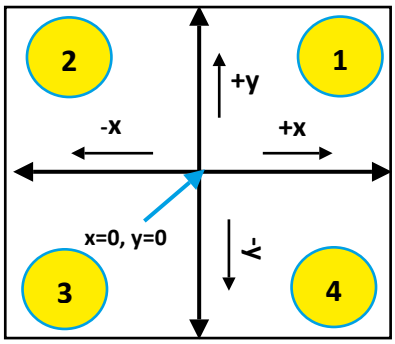
```
import turtle

#Setup turtle screen
stage = turtle.Screen()
stage.bgcolor("black")
stage.title("Welcome to my game")

#Create player turtle
fred = turtle.Turtle()
fred.color("white")
fred.shape("turtle")

fred.forward(100)
```

NEW LINES of code: Changing the colour and shape of the *player turtle, Fred*

Now that we have played around with Python Turtle a bit, let us start building our game.

We have our ***turtle window*** and ***player turtle*** set up. Let us now look at defining a game area for the ***player turtle*** to move around in. We will do this by creating another 'turtle' to control and call this, say, ***border*** (our ***boundary turtle***).

Any position in the turtle window can be specified using **x (horizontal) and y (vertical) coordinates**. For example, any turtle created is always placed at the centre of the turtle window, at position **x=0** and **y=0**, as you can see is true for our ***player turtle, Fred***. Based on this, the turtle window can be divided into four quadrants, as shown in the figure.



Let us choose the starting point of our boundary to be in the 3rd quadrant i.e., with **-x and -y coordinates**. For example, (-250, -250).

Now, let us look at what the above looks like in code. Add the following lines of code **before** the lines for creating the ***player turtle***.

```
#Draw the game area
border = turtle.Turtle()
border.color("yellow")
border.penup()
border.setposition(-250,-250)
border.pendown()
border.pensize(3)
```

`penup()` lets the border turtle move without drawing any lines.
`setposition()` moves the border turtle to the 3rd quadrant to start drawing the game area.
`pendown()` lets the turtle draw lines again.

CREATING THE GAME AREA

Continuing on from the previous code where we had started drawing the game area, let us say that we would like the area to be in the shape of a square.

To do that you will need to move the ***border turtle*** forward, turn left, go forward again and so on. Add the following lines to make this happen:

```
border.forward(500)
border.left(90)
border.forward(500)
border.left(90)
border.forward(500)
border.left(90)
border.forward(500)
border.left(90)
```

The idea is to draw a line from x=−250 to x=250. So, the player turtle goes from x=−250 to x=0 i.e., 250 pixels and then from x=0 to x=250 i.e., again 250 pixels, giving a total distance of 500 pixels.

Run the code and you should see the game area being created with the ***player turtle*** in the centre.

Now, the square that you just made had the 'forward' and 'left' movements being repeated 4 times. **Repetition** or **Iteration** in code can be put in a **loop**.

There are different kinds of loops. We will be using the **'for' loop** here. Replace the above lines with the following lines:

```
for i in range(4):
    border.forward(500)
    border.left(90)
```

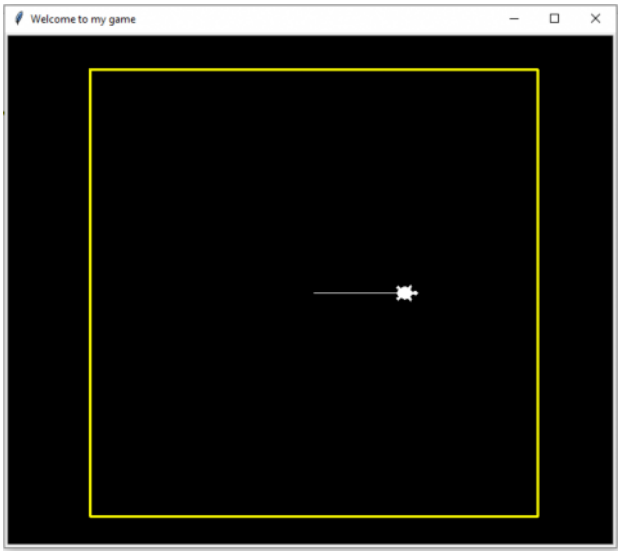
Note the indentation: this is added automatically once you press ENTER after the line with the 'for' instruction.

Finally, once the game area has been drawn, we don't really want to see the border turtle. So, we hide it using the following line:

```
border.hideturtle()
```

The final code and turtle output window look like this:

```
1  import turtle
2
3  #Setup turtle screen
4  stage = turtle.Screen()
5  stage.bgcolor("black")
6  stage.title("Welcome to my game")
7
8  #Draw the game area
9  border = turtle.Turtle()
10 border.color("yellow")
11 border.penup()
12 border.setposition(-250,-250)
13 border.pendown()
14 border.pensize(3)
15 for i in range(4):
16     border.forward(500)
17     border.left(90)
18 border.hideturtle()
19
20 #Create player turtle
21 fred = turtle.Turtle()
22 fred.color("white")
23 fred.shape("turtle")
24
25 fred.forward(100)
```



MOVE THE PLAYER

Now that we have game area and player turtle set up, let us look at how to get the turtle to move in response to key presses.

We want the ***player turtle*** to move up, down, left, right in the game area. We can easily link these movements to the corresponding arrow keys (direction keys) on your keyboard using the Python Turtle library.

We will be using the **`listen()`** and **`onkey()`** from the Python Turtle library. Both these functions are available as part of the ***Turtle Screen***. The **`listen()`** detects if a key has been pressed and **`onkey()`** lets you bind an 'action' based on what key is being pressed.

To access these functions, we will be using the ***Turtle Screen object***, ***stage***, that we created at the beginning.

Add the following lines to your existing program:

```
#Set keyboard bindings
stage.listen()
stage.onkey(go_right, "Right")
stage.onkey(go_up, "Up")
stage.onkey(go_down, "Down")
stage.onkey(go_left, "Left")
```

go_right, go_left, go_up, and go_down are functions that make the player turtle move in response to the key pressed.
A function is a set of instructions which only runs when you 'call' it in the program, as we have done here.

Direction key on the keyboard

Our next task is to define the functions that are being called in the above lines. This needs to be done before the functions are called. So, type in the following new lines **before** the above lines.

```
#Functions to bind to key presses -
#controlling the movement of the player
def go_up():
    fred.setheading(90)
def go_down():
    fred.setheading(270)
def go_left():
    fred.setheading(180)
def go_right():
    fred.setheading(0)
```

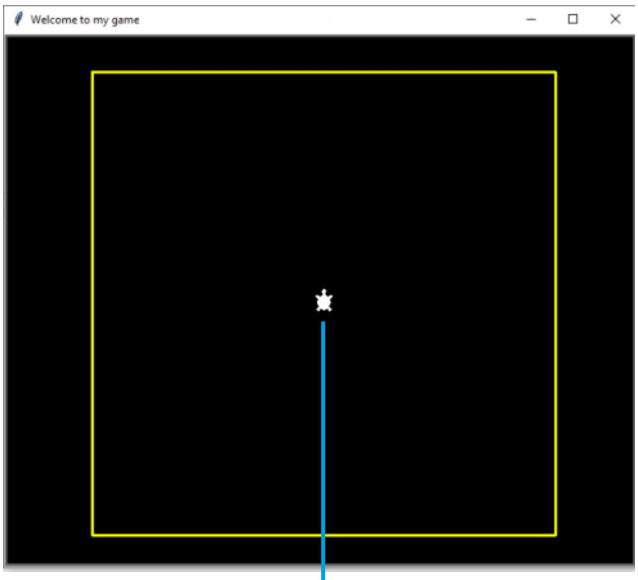
Sets the orientation of the *player turtle* to the angle within the `()`.
For example, `fred.setheading(90)` changes the orientation of the turtle to north.

MOVE THE PLAYER

After adding the code for detecting a key press (as we did on the previous page), this is what part of the code editor and turtle window look like:

```
#Create player turtle
fred = turtle.Turtle()
fred.color("white")
fred.shape("turtle")

#Functions to bind to key presses -
#controlling the movement of the pla
def go_up():
    fred.setheading(90)

def go_down():
    fred.setheading(270)

def go_left():
    fred.setheading(180)

def go_right():
    fred.setheading(0)

#Set keyboard bindings
stage.listen()
stage.onkey(go_right, "Right")
stage.onkey(go_up, "Up")
stage.onkey(go_down, "Down")
stage.onkey(go_left, "Left")
```

Note the `fred.forward(100)` line has been removed.

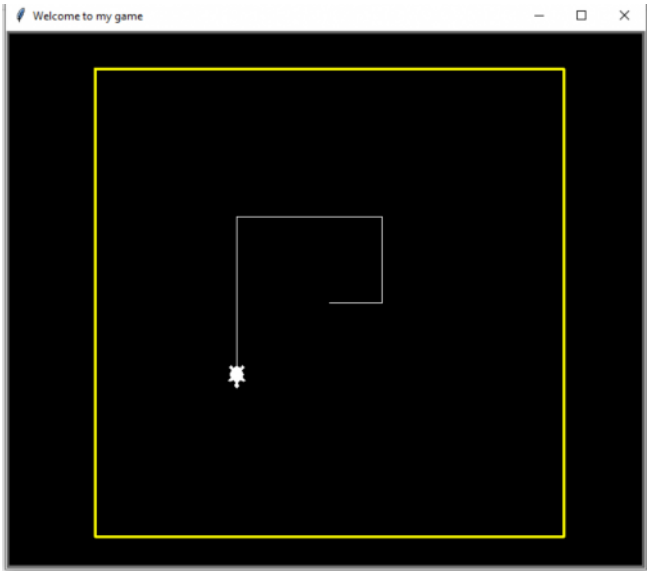


The player turtle turns to face north when the 'up' key is pressed.

Next, we need the ***player turtle*** to keep moving forward by 1 step in the direction it is facing. To do this add the following lines to your existing code:

```
while True:
    fred.forward(1)
```

Press **F5** to run the program. The ***player turtle*** should be now moving around in the game area based on key presses.



COLLISION CHECK

OK, the game's nearly ready!

Next, we need to check for collisions, either with the boundaries of the game area or with paths that are already drawn by the ***player turtle***.

Let us start with checking collisions at the x and y boundaries. We know the four corners of the game area have the following coordinates **(x,y)**: **(-250,-250)**, **(250, -250)**, **(250, 250)** and **(-250, 250)**. Therefore, these coordinates define the boundaries we need to check. So, if the ***player turtle*** goes beyond either +250 or -250 on the x-coordinate or +250 or -250 on the y-coordinate, the game should stop.

Add the boundary check within the **`while True`** loop using the following lines:

```
#Boundary checking for turtle - xcor
if fred.xcor() > 240 or fred.xcor() < -240:
        break;
#Boundary checking for turtle - ycor
if fred.ycor() > 240 or fred.ycor() < -240:
        break;
```

Note the test boundary values used are slightly less than the actual boundary values of +/-250.
What happens if you use actual boundary values here instead?

Run the program and make sure this check works.

Our next task is to detect collisions between paths taken by the ***player turtle***. Any overlap should lead to the game ending.

For this, we are going to make use of **Python lists**. **A list is a collection of data that can be amended as the program runs.**

Our Python list is going to contain the x and y coordinates of all the positions of our player turtle as it moves around the game area. The current position of the player turtle is then checked against this list and if the position exists, there must be an overlap with another path and, therefore, the game ends.

```
#Recording turtle's  positions to check for overlap
positions = []

#Overlap check
position = (fred.xcor(), fred.ycor())

if position in positions:
    break;
else:
    positions.append(position)
```

Empty Python list to store the different positions of the *player turtle*.
NOTE: Add this line just after you create the *player turtle*.

NOTE: Add these lines within the 'while True' loop.

Record the current position of the *player turtle*.

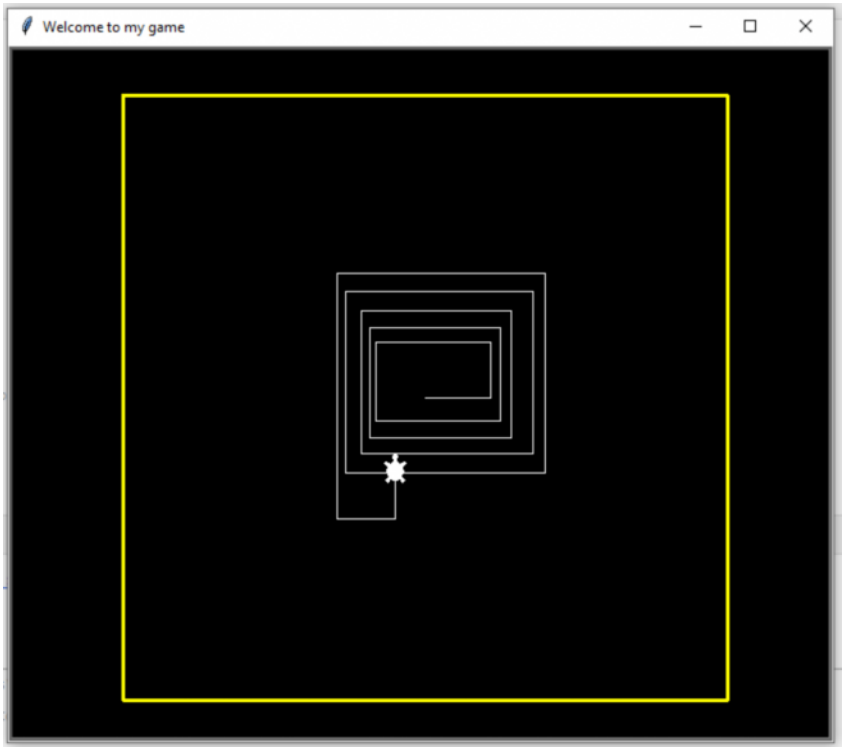
If the current position of the player is in the positions list, end the game. Else, add the new position to the list.

TRON GAME REMIX

That's it. Your game's ready!

This is what the final code and output look like:

```
import turtle
#Setup turtle screen
stage = turtle.Screen()
stage.bgcolor("black")
stage.title("Welcome to my game")

#Draw the game area
border = turtle.Turtle()
border.color("yellow")
border.penup()
border.setposition(-250,-250)
border.pendown()
border.pensize(3)
for i in range(4):
    border.forward(500)
    border.left(90)
border.hideturtle()

#Create player turtle
fred = turtle.Turtle()
fred.color("white")
fred.shape("turtle")

#Recording turtle's positions to ch
positions = []

#Functions to bind to key presses -
#controlling the movement of the pl
def go_up():
    fred.setheading(90)

def go_down():
    fred.setheading(270)

def go_left():
    fred.setheading(180)

def go_right():
    fred.setheading(0)

#Set keyboard bindings
stage.listen()
stage.onkey(go_right, "Right")
stage.onkey(go_up, "Up")
stage.onkey(go_down, "Down")
stage.onkey(go_left, "Left")
```




```
while True:
    fred.forward(1)

    #Boundary checking for turtle - xcor
    if fred.xcor() > 240 or fred.xcor() < -240:
        break;
    #Boundary checking for turtle - ycor
    if fred.ycor() > 240 or fred.ycor() < -240:
        break;

     #Overlap check
    position = (fred.xcor(), fred.ycor())

    if position in positions:
        break;
    else:
        positions.append(position)
```

EXTENSION ACTIVITY - DISPLAYING A MESSAGE

Now, for some additions to the Turtle Window display.

Even though the game ends as we want it to, there is no message that comes up on the turtle window.

To write within the turtle window, let us create a new turtle, ***pen***.

Create the new turtle where the other turtles have been created:

```
#Create pen turtle
pen =  turtle.Turtle()
pen.penup()
pen.color("red")
pen.hideturtle()
```

Next, we can create a function called **`end_game()`** and use the **`write()`** from the Turtle library to write a message on the Turtle window.

Define **`end_game()`** where the other functions have been defined and add these lines as shown below:

```
#Function to end the game
def end_game():
    pen.setposition(-330, -240)
    pen.write("Game over", True, align="left", font
                                    =("Arial", 14, "bold")
```

This positions the pen turtle at the bottom left-hand corner of the screen. You can choose where you want to show messages.

Now, add a call to **`end_game()`** within the **`while True`** loop every time a collision happens or the ***player turtle*** touches the boundaries of the game area.

This is how to add the above lines to your existing code:

```
46  def go_left():
47      fred.setheading(180)
48
49  def go_right():
50      fred.setheading(0)
51
52  #Function to end the game
53  def end_game():
54      pen.setposition(-330,-240)
55      pen.write("Game over", True, align="left", font=("Arial", 14, "bold"))
56
```

Defining `end_game()`

Calling `end_game()` every time a collision happens or the *player turtle* crosses the game area boundaries.

```
69  while True:
70      fred.forward(1)
71
72      #Boundary checking for turtle - xcor
73      if fred.xcor() > 240 or fred.xcor() < -240:
74          end_game()
75          break;
76      #Boundary checking for turtle - ycor
77      if fred.ycor() > 240 or fred.ycor() < -240:
78          end_game()
79          break;
80
81       #Overlap check
82      position = (fred.xcor(), fred.ycor())
83
84      if position in positions:
85          end_game()
86          break;
87      else:
88          positions.append(position)
```

EXTENSION ACTIVITY - DISPLAYING END TIME

We can also display how long the game went on for. This will show you how well you played!

There's a few lines to add, so just look at the code snippet images below and add the new lines in the appropriate locations.

We start by importing the time library.

```
import time
```

```
1  import turtle
2  import time
3
4  #Setup turtle screen
5  stage = turtle.Screen()
```

We store the game's start time in a variable.

```
startTime = time.time()
```

```
33 #Recording turtle's positions to check for overlap
34 positions = []
35
36 #Use a variable to store the number of seconds since epoch.
37 #Epoch is where time is taken to begin and this is
38 #January 1, 1970 00:00:00 UTC
39 startTime = time.time()
40
```

Next is to define a function to display the total time that the player had the game running for. Basically, tells you how expertly you controlled the turtle player!

```
def show_time():
    pen.setposition(-330,260)
    pen.write(str(round(t, 2)) + " secs", True, align="left",
          font=("Arial", 14, "bold"))
```

Here, `t` is the time that you kept the turtle going for. `t` is calculated in the '`while True`' loop, as shown below. Also, useful to '`round`' off `t` to 2 decimal places.

```
56 #Function to end the game
57 def end_game():
58     pen.setposition(-330,-240)
59     pen.write("Game over", True, align="left", font=("Arial", 14, "bold"))
60
61 #Function to show how much time the game was running for
62 def show_time():
63     pen.setposition(-330,260)
64     pen.write(str(round(t, 2)) + " secs", True, align="left", font=("Arial", 14, "bold"))
```

Next, within the **`while True`** loop, calculate the game time duration.

Finally, add a call to **`show_time()`** within the **`while True`** loop every time the **`end_game()`** function is called.

```
72 while True:
73
74     fred.forward(1)
75
76     t = time.time()-startTime
77
78     #Boundary checking for turtle - xcor
79     if fred.xcor() > 240 or fred.xcor() < -240:
80         end_game()
81         show_time()
82         break;
83     #Boundary checking for turtle - ycor
84     if fred.ycor() > 240 or fred.ycor() < -240:
85         end_game()
86         show_time()
87         break;
88
89      #Overlap check
90     position = (fred.xcor(), fred.ycor())
91
92     if position in positions:
93         end_game()
94         show_time()
95         break;
96     else:
97         positions.append(position)
98
```