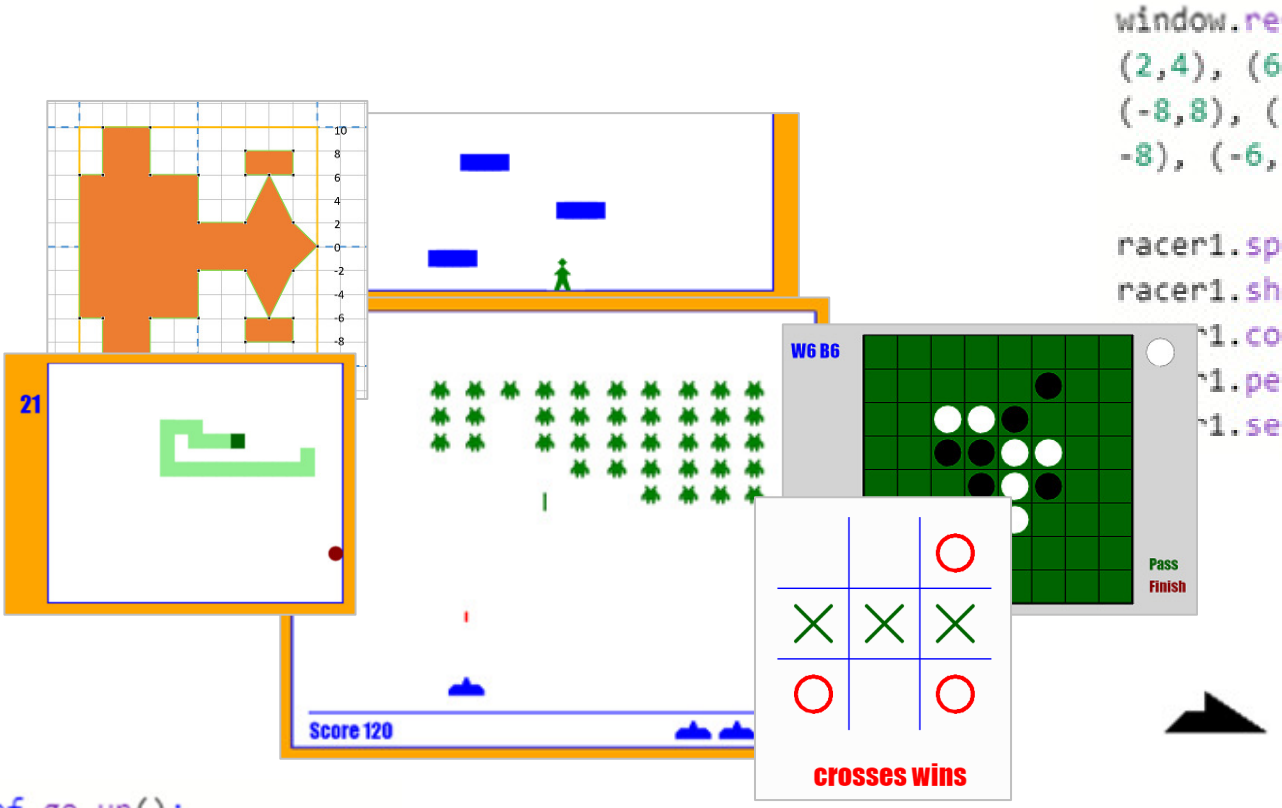
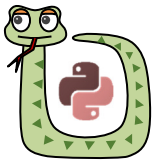
Python Games




```
window.register
(2,4), (6,6), (
(-8,8), (-8,4),
-8), (-6,-10)))

racer1.speed(0)
racer1.shape('r
1.color("d
1.penup()
1.setposit
```

```
def go_up():

    ypos = ship.ycor()

    if ypos <= 230:
        setheading(90)
```

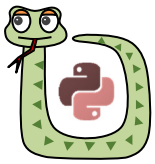
Dan Collingbourne



Skills

Games

Difficulty



Your challenge is to create some games using the Python programming language. Although it's not absolutely necessary, it will be helpful if you have some prior coding experience. We will be working through the solutions step by step but also progressing quite quickly. The idea is that you select a game to build then study the associated skills tasks before tackling it.

Aim: *To learn how to create, debug and fork programs using Python (with turtle).*

Note: *For a gentler introduction to Python programming please see our publication 'Python Graphics'.*

Task 1 – repl.it

We're going to create our solutions in a free website called *repl.it*. This site allows you to play around with bits of Python code without the need to install applications. You may use repl.it for your solutions or you might work in a *Python (with Turtle)* package installed on your school computers. The benefit of using repl.it is that you can work at home, school or anywhere else with an internet connection.

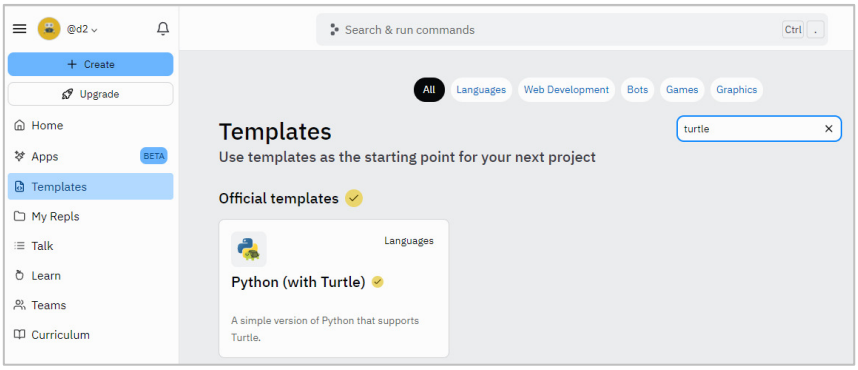
If you are not using repl.it please jump to the next task. For those using repl.it, continue below.

a. Navigate to the website https://www.repl.it/ and follow the instructions to set up a free account. We suggest you don't add personal information such as your real name. You should definitely avoid posting photographs.

b. Once logged in, click on the *Templates* link in the menu on the left.

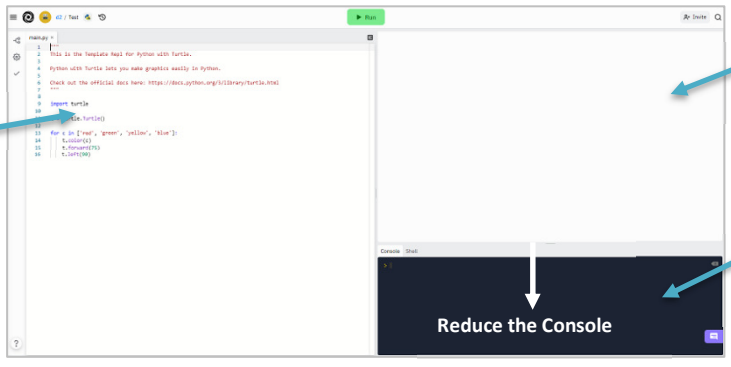
c. We want to start with the *Python (with Turtle)* template. If the template is visible, simply select it. If it is not visible then search for *'turtle'* and the template should appear as the only result.

Note: *Don't open a Python template; we need 'Python (with Turtle)'.*



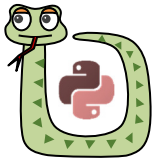
d. Give your new *repl* the name '*Test*' and click the *Create Repl* button. You will be taken to the coding page shown below.

The left half of the screen is where you type your code. This is the called the *input*.



The top right section is where the results are displayed. This is the *output*.

The bottom right section is called the *console*. It displays errors and other messages. You may reduce the size of the console by dragging down the bar above.



In the games created here, the movements of an object will be controlled using the keyboard or mouse. Task 1 below is also very important as it shows how to set up and call functions, used in all but the simplest of Python programs.

Aim: *Learn how to control the turtle using the keyboard and mouse.*

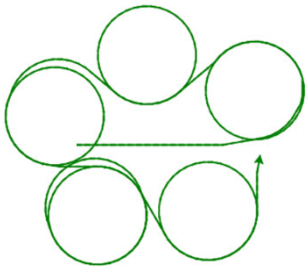
Task 1 – Keyboard Commands

The program on the right will give you keyboard control over a Turtle object called *scribe*. The keyboard commands are as follows:

- ***p*** *Move forward 10 pixels (hold down to repeat)*
- ***w*** *Turn right 10°*
- ***q*** *Turn left 10°*

Create the program and use the controls to produce a pattern. Our picture below was an attempt at drawing crop circles. Save as "**A2.1 Keyboard Commands**".

Note: *You might have to click on the output window to make the controls active.*



```
1   import turtle
2
3   #DEFINE FUNCTIONS
4
5   def go_forward():
6     scribe.forward(10)
7
8   def go_left():
9     scribe.left(10)
10
11  def go_right():
12    scribe.right(10)
13
14
15  #ACTUAL PROGRAM
16
17  scribe = turtle.Turtle()
18  window = turtle.Screen()
19
20  scribe.color("green")
21  scribe.pensize(2)
22
23  window.onkey(go_forward,"p")
24  window.onkey(go_right,"w")
25  window.onkey(go_left,"q")
26  window.listen()
```

The lines within the function must be indented. Use a single Tab indent or two spaces.

Calling the function takes the program flow through the function before returning to the original point.

Questions about the program

a. Functions are separate sections of programming code that perform a specific task. Three functions have been defined at the start of the program above. One of these is called *go_forward*. What are the other two called?

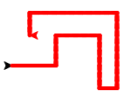
b. What name is given to the first *Turtle* object created in this program?

c. The *Screen* object must be defined before our keyboard commands can be picked up. What name is given to this object and on which line is it defined?

d. The *listen* method tells the program to expect user input (the program is waiting for events). On which line is this initiated?

e. The program is waiting for *events* such as the keyboard strokes p, w and q. On which three lines are these events set up?

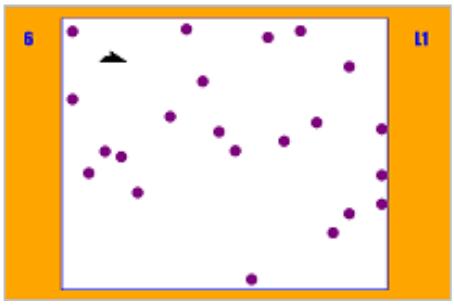
f. The *go_forward* function is called in line 23. What event causes this to happen?



Edit the program in order to create some different style patterns.

Difficulty = 1; Lines of code in solution = 223.

This game involves dodging objects that are moving across the screen. The basic game is reasonably straightforward but the possibilities for extension are endless.



	Section	Req
A1	Getting Started	✓
A2	Functions & Controls	✓
A3	Playing Areas	✓
A4	Counting & Displays	?
A5	Inputs & Data Validation	X
A6	Lists	✓

Required? ✓=yes, X=no, ?=possibly

Task 1 – The Basic Game

Setting up the Screen

e. Fork your solution to "**A3.1 Playing Space**" from the *Playing Areas* tasks, naming the new copy "**G2 Dodge 1**".

f. Remove the sections of code that create the dashed lines. These are not needed.

g. Define another turtle called *ship*. This line of code can be placed at the top of the program so that the ship object can be accessed everywhere (i.e. it is global).

```
2   window = turtle.Screen()
3
4   ship = turtle.Turtle()
```

h. For now, design the ship as a simple black square placed inside the left border, in the middle vertically. Place this code in its own function.

i. Call the function after you have set up the playing area.

j. Add functions to control the up and down movements, triggered by pressing the up and down arrow keys. This code can be typed or copied from your solution to "**A2.2 Arrow Keys**" (this program could be opened in a second browser window). Remember to change the name of your turtle to *ship* in the new code.

```
def design_ship():
    ship.speed(0)
    ship.shape("square")
    ship.color("black")
    ship.penup()
    ship.setposition(-200,0)


#ACTUAL PROGRAM

setup()
design_ship()
```

Adding the Projectiles

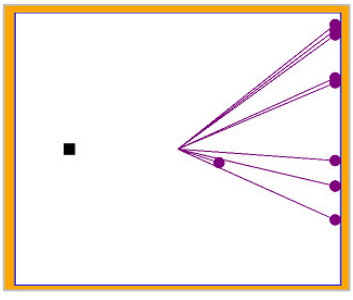
It's now time to add some projectiles that will move along the screen from right to left. Each projectile will be a new turtle, beginning its journey from a random spot on the right-hand side. The turtles will be stored in a list.

Note: *Our partially obscured solution is shown on the next page but try this coding for yourself first.*

a. If you haven't already, work through the first section of the *Counting & Displays* resource looking at random numbers. Import the *random* module into your program.

b. At the end of your code, define an empty list for your turtles called *turtle_list*.

c. Create a loop which will run until we tell it to stop. Within this loop, create an *if* statement that adds a new turtle to the list but only if there are less than 20 already. Use the *len* function to count the number of turtles in the list (see below right).



```
end_game = 0

while end_game == 0:
```

```
number_of_turtles = len(turtle_list)
```

Dodge (page 4)

Task 2 – Fixing the Problems

We found the game worked reasonably well at this point, although we made the following observations:

- The ship could move off the top and bottom of the screen;
- There was a flicker in the centre of the screen when each turtle was created;
- The game was too easy;
- Whilst experimenting with different values to make the game harder, we experienced the *out of range* error shown on the right.

```
Console  Shell
IndexError: list index out of range on line 103
>
```

Fork the program, name the copy "**G2 Dodge 2**" and have a go at solving some of these problems yourself. If you need help then work through the rest of this section.

a. Limitations can be put on the ship's movement using code such as ours on the right.

```
def go_up():

  ypos = ship.ycor()

  if ypos <= 230:
    ship.setheading(90)
    ship.forward(20)
```

b. The flickering might be controlled by changing the order of events when a new projectile is created.

```
    turtle_list[nexti].speed(0)
    turtle_list[nexti].penup()
    turtle_list[nexti].setposition(290,
    turtle_list[nexti].color("purple")
```

If the above trick doesn't work for you, try the *tracer* and *update* methods as used on the right. These will turn off all animation whilst the turtle is being created. However, a side effect of this is that it might also speed up the game significantly. If so, see the solutions below.

```
window.tracer(0, 0)
turtle_list.append(turtle.Turtle())   #add a turtle
turtle_list[nexti].speed(0)
turtle_list[nexti].penup()
turtle_list[nexti].setposition(290, random_y)
turtle_list[nexti].color("purple")
turtle_list[nexti].shape("circle")
turtle_list[nexti].setheading(180)   #point right
window.tracer(1)
window.update()
```

c. How difficult your game is depends on the platform you are using (i.e. the repl.it website or some other installation). Even if you are using repl.it, there might be huge differences in the speed of movement depending on the template you started with. Finally, turning off the animation using the *tracer* method could speed things up tremendously.

The ideas below will help you control how difficult the game is.

- Try changing speed with which the projectiles move across the screen by editing the *forward* jump size.

```
turtle_list[i].forward(30)
```

Task 3 – Adding Functionality

There are lots of ways you might improve your Dodge game. In a fork your last solution named "**G2 Dodge 3**" try some of the following.

Game Over Display

Display the words "GAME OVER" when the ship is hit by a projectile. We have placed the code in a function called *game_over*.

```
if turtle_distance < 20:
    ship.color("red")
    game_over()
```

Start Display

Creating a *Start* event is a little more difficult. The main program should be placed inside a function so that it can be called later (this is good practice anyway).

```
def run_program():

    window.onkey(go_up,"up")
    window.onkey(go_down,"down")

    window.listen()
```

a. Create a function for your main program. Every line moved into the function will need one further indent.

b. Create another function which displays a start message. Choose a key to run the main function when it is pressed.

Note: *You will need to define the turtle globally as it will be needed from other functions.*

```
def wait_start():
    StartDisplay.speed(0)
    StartDisplay.color("red")
    StartDisplay.penup()
    StartDisplay.setposition(-17
    StartDisplay.write("Press th
    StartDisplay.hideturtle()

    window.onkey(run_program,"do
    window.listen()
```

c. Hide the start display when the main program runs.

```
#Hide the Start Display
StartDisplay.clear()
```

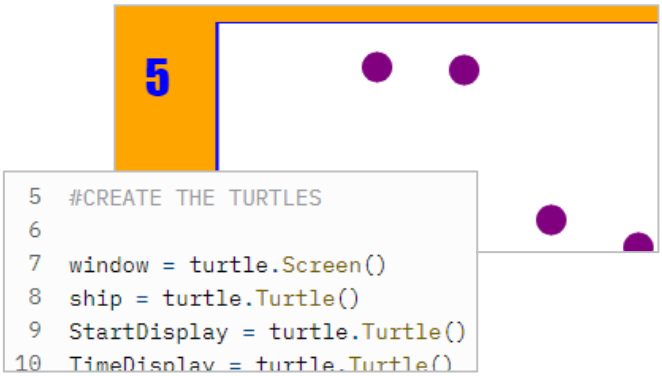
d. Your actual program should now set up the page and run the start function.

Timer

Add a timer to your game. If you haven't already, look through the resource *Counting & Displays* and reuse the code from that program.

Find a suitable place in your loop to recalculate and update the time. This shouldn't happen too often as it will slow down the action.

Remember to import the *time* module and define turtles globally if they are needed in different functions.



Task 3 – Adding Functionality (cont.)

Stopping the Screen Refresh

You might decide that things are still all moving a little too slowly. The problem is that the screen is refreshing each time one of the projectiles moves a step to the left; this is slowing everything down. To speed up the gameplay (and smooth the visual appearance) we can refresh the display only after all the projectiles have been moved on a step. Use the code on the right before and after the loop to put this into place.

```
window.tracer(0, 0)

for i in range(number_of_turt
```

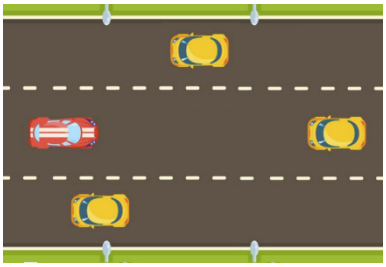
Note: *We have also shifted the calling of our game_over function so that the last screen refresh takes place before everything stops. Without this edit, the ship doesn't appear to have been hit by a projectile.*

```
        ship.color("red")
        end_game = 1

    window.tracer(1)
    window.update()

    if end_game == 1:
      game_over()
```

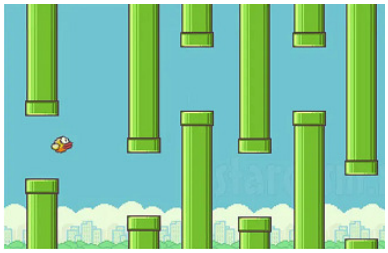
Extension – Super Challenge

Through the years, there have been lots of games that involve objects travelling across the screen that must be avoided. Develop your game in any way that you choose. There are some ideas below.

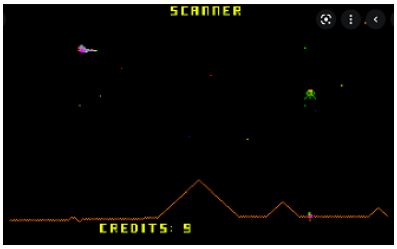
When building your game, break the development down into a series of small steps to take. Although this isn't really how you would develop a game professionally, it's a good way for a beginner to learn. You will often find that the solutions are easier to find than you thought they might be.



Car Racing



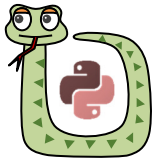
Flappy Bird



Defender

Notes:

- If you are using replit, you won't be able to import images to use in your game. You will therefore need to create shapes yourself to use. Keep things visually simple and concentrate on the functionality.
- Try and make a scrolling game infinite by generating objects randomly rather than designing them precisely.
- The Defender game has a spaceship that can shoot a laser. Our Space Invaders game will help with this functionality.



Answers

Task 1 – The Basic Game

Questions about the Program

1. We have used the variable *nexti* for the next index in the list. Why is this equal to the number of turtles?

   If there are 10 turtles, their indices will be 0-9. The next index is therefore 10.

2. Does the program stop running when 20 turtles have been created?

   No. The program is still cycling the loop but no more projectiles are created. You can still move the ship.

3. We want the turtles to be formed instantly on the right-hand side without leaving a trace from the centre. How could this be achieved?

```
turtle_list[nexti].speed(0)
turtle_list[nexti].penup()
```

4. We then want the turtles to gradually move to the left. How could this be programmed?

   As per solution.

More Questions about the Program

1. What code adds 1 to the turtle count after a new one has been created?

```
number_of_turtles = number_of_turtles + 1
```

2. When moving through the list of turtles, why have we used the code *range(number_of_turtles-1)*?

   We are working with indices which start from 0. If there are 5 turtles, they will have the indices 0, 1, 2, 3 and 4.

3. How could you hide the turtles when they reach the lefthand wall?

```
turtle_list[i].hideturtle()
```

   They can also be deleted as they are no longer needed.

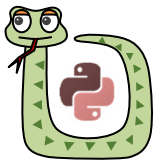
```
del turtle_list[i]
```

4. How do you think we'll test to see if our ship has been hit by a projectile?

   Measure the distance between the ship and each turtle. Act if this distance is less than the one specified.

```
turtle_distance = turtle_list[i].distance(ship)

if turtle_distance < 20:
```

Dodge Answers (page 2)

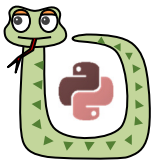


Task 1 – The Basic Game (cont.)

https://replit.com/@d2/G2-Dodge-1

```
import turtle
import random

#CREATE THE TURTLES

window = turtle.Screen()
ship = turtle.Turtle()

#DEFINE FUNCTIONS

#Set up the screen
def setup():
  window.bgcolor("orange")

  Playing = turtle.Turtle()
  Playing.color("blue")
  Playing.pensize(2)
  Playing.speed(0)
  Playing.penup()
  Playing.setposition(300,250)
  Playing.pendown()
  Playing.fillcolor("white")
  Playing.begin_fill()
  Playing.setposition(300,-250)
  Playing.setposition(-300,-250)
  Playing.setposition(-300,250)
  Playing.setposition(300,250)
  Playing.end_fill()
  Playing.hideturtle()


#Design our ship

def design_ship():
  ship.speed(0)
  ship.shape("square")
  ship.color("black")
  ship.penup()
  ship.setposition(-200,0)
```

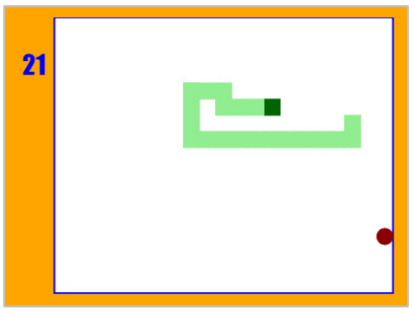
```
#Directional control

def go_up():
  ship.setheading(90)
  ship.forward(20)


def go_down():
  ship.setheading(270)
  ship.forward(20)


#ACTUAL PROGRAM

setup()
design_ship()

window.onkey(go_up,"Up")
window.onkey(go_down,"Down")

window.listen()


            #Continued next page
```



Difficulty = 2; Lines of code in solution = 255.

This game involves steering a snake without hitting the walls or its own tail. Food items are gathered by the snake, each making it a little longer.

If you're not familiar with the game, have a look for a version online to play. There is one hidden in Google Search.



	Section	Req
A1	Getting Started	✓
A2	Functions & Controls	✓
A3	Playing Areas	✓
A4	Counting & Displays	?
A5	Inputs & Data Validation	X
A6	Lists	✓

Required? ✓=yes, X=no, ?=possibly

Task 1 – The Basic Game

Setting up the Playing Area

k. Fork your solution to "**A3.1 Playing Space**" from the *Playing Areas* tasks, naming the new copy "**G3 Snake 1**".

Note: *At the time of writing, this game had serious issues when starting with the repl.it template. We suggest you fork our plain template and copy your code into this.* https://replit.com/@d2/Template

l. Remove the sections of code that create the dashed lines. These are not needed.

m. We will look at our playing space as a set of small squares, each 20 x 20 pixels. The turtle head will move from square to square and the body will follow.

Edit your playing space so that the x values now range from -310 to 310 and the y values from -270 to 270. The centres of the squares have coordinates such as those shown below. You might need to adapt this design for a smaller screen size.

Corner
x -310, y 270

x -300 y 260	x -280 y 260	x -260 y 260	x -240 y 260	x -220 y 260	x -200 y 260	x -180 y 260	x -160 y 260	x -140 y 260	x -120 y 260	x -100 y 260	x -80 y 260	x -60 y 260
x -300 y 240	x -280 y 240	x -260 y 240	x -240 y 240	x -220 y 240	x -200 y 240	x -180 y 240	x -160 y 240	x -140 y 240	x -120 y 240	x -100 y 240	x -80 y 240	x -60 y 240
x -300 y 220	x -280 y 220	x -260 y 220	x -240 y 220	x -220 y 220	x -200 y 220	x -180 y 220	x -160 y 220	x -140 y 220	x -120 y 220	x -100 y 220	x -80 y 220	x -60 y 220

The Snake Head

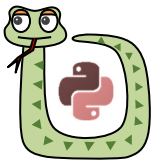
a. Define another turtle called *head*. This line of code can be placed at the top of the program so that the object can be accessed everywhere (i.e. it is global).

```
4   #CREATE THE TURTLES
5   window = turtle.Screen()
6   head = turtle.Turtle()
```

b. Design the snake as a simple square of your chosen colour. By default, it will be positioned in the centre of the screen. Place this code in its own function.

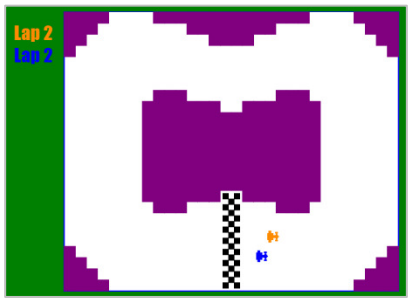
c. Call the function after you have set up the playing area.

```
def snake_head():
    head.shape("squ
    head.color("dar
    head.penup()


#ACTUAL PROGRAM
setup()
snake_head()
```



Difficulty = 3; Lines of code in solution = 412.

Design some racers to race around a track then take this game forward with your own bells and whistles.



	Section	Req
A1	Getting Started	✓
A2	Functions & Controls	✓
A3	Playing Areas	✓
A4	Counting & Displays	✓
A5	Inputs & Data Validation	?
A6	Lists	✓

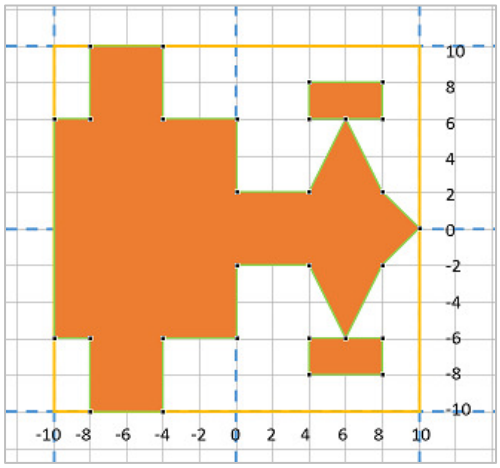
Required? ✓=yes, X=no, ?=possibly

Task 1 – The Basic Setup

Designing your Racer

n. Fork your solution to "**A3.1 Playing Space**" from the *Playing Areas* tasks, naming the new copy "**G6 Race 1**". Remove the sections of code that create the dashed lines; these are not needed.

Note: *At the time of writing, this game had serious issues when starting with the repl.it template. We suggest you fork our template and copy your code into this.* https://replit.com/@d2/Template



o. Design a racer for your game. We are going to use cars but you can race whatever you like. Create your racer so that it occupies a space up to 20 x 20 pixels, centred on (0,0) and pointing to the right.

You may generate coordinates for the vertices any way you like; we designed our character using the tools in Microsoft Word:

1. Open a new Word document and select '**Layout / Align / View Gridlines**'.
2. Click '**Layout / Align / Grid Settings**' and choose a horizontal and vertical spacing of 1cm. Click *OK*.
3. Select '**Insert / Shapes / Freeform: Shape**'.
4. Click on the grid axes repeatedly to create the shape. Finish with a double click.
5. To edit the shape, right click on it and select *Edit Points*. You may then move the vertices. You may also add and delete points.

p. Once you have a set of coordinate pairs ready, create your first racer using code such as ours on the right. Define your racer globally (at the top of your program) and place the rest of the code at the end of your setup function.

The coordinates are listed in turn using the form (y,x), each separated by a comma. Let the coordinate pairs run onto new lines as you type rather than forcing new lines. It doesn't matter where in your shape you start.

```
window.register_shape("racer", ((6,-10), (6,-8),
(2,4), (6,6), (6,4), (8,4), (8,8), (6,8), (6,6),
(-8,8), (-8,4), (-6,4), (-6,6), (-2,4), (-2,0),
-8), (-6,-10)))

racer1.speed(0)
racer1.shape('racer')
racer1.color("darkorange")
racer1.penup()
racer1.setposition(0,-150)
```

Task 1 – The Basic Setup (cont.)

Moving your Racer

Use the code from the *Keyboard Commands* task in the *Functions & Controls* resource to move your racer forwards. You will need to set up the *onkey* functions and tell your window to listen for events.

```
def go_left1():
    racer1.left(10)

def go_right1():
    racer1.right(10

def go_forward1()
```

Hitting the Boundaries

Add some *if* conditions so that the racer can't leave the boundaries of the playing area. If it does, then move the racer backwards (perhaps 30 pixels?).

```
    racer1.forward(10)

    xpos = racer1.xcor()
    ypos = racer1.ycor()

    if xpos > 290 or xpos < -290 or yp
        racer1.backward(30)
```

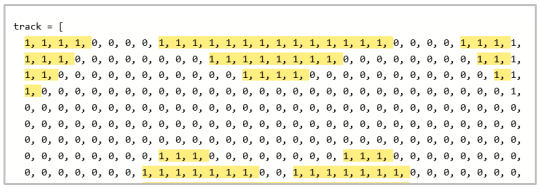
The Track

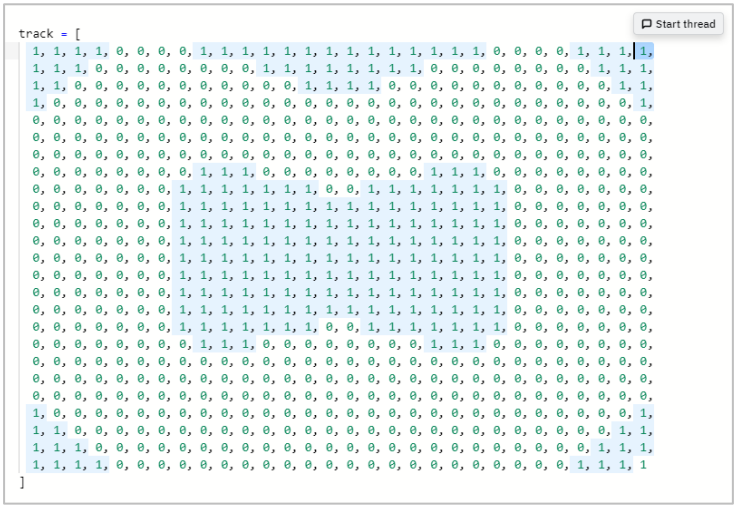
It's time to design your track. In the main area of your program, define a list for the obstacles. The list should contain a 1 where you want to place a block and a 0 otherwise.

Thankfully, you may use the *Enter* key within the list without affecting the program, so make the list 30 items long and 25 items down (as ours below right). You may need to adapt the track size if you are working on a small screen.

```
track = [
  1, 1, 1, 1, 0, 0,
  1, 1, 1, 0, 0, 0,
  1, 1, 0, 0, 0, 0,
  1, 0, 0, 0, 0, 0,
```

Note: *If using repl.it, you may be able to select a single 1 anywhere in your list and the webpage will highlight all the others. This makes track design easy. If this functionality is not available you may create the list in MS Word and search for the '1, 's instead.*



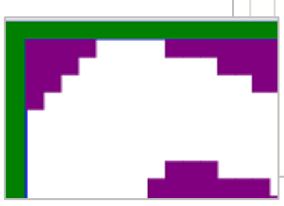


Have a go at printing the track to the screen. There are a few ways of doing this but we have used a simple loop, part of which is shown on the right.

Our turtle, called *obstacle*, moves from the left to the right of the screen, top to bottom. Each jump is 20 pixels. We are also moving through the track list one step at a time. If the list element contains a 1 we stamp a square on the screen.

Continue to the next page when you have had a go at completing this coding yourself.

```
for i in range(len(track)):

    obstacle.setposition(x, y)

    block = track[i]

    if block == 1:
        obstacle.stamp()

    x = x + 20
```



Task 2 – Fixing the Problems (cont.)

- One way of solving this might involve creating a list of invaders to delete in the *shoot* function (you can still hide the turtles without deleting them) then deleting all those in the list in the *move_invaders* function. You should then redefine the blank list. This is likely to lead to different issues, however;
- Another approach might involve having a defined set of locations at any point and aligning the invaders with these.

Task 3 – Adding Functionality

We'll now look at adding further functionality. Fork your last solution and name the copy "**G7 Invaders 3**".

Start Display

a. Create a function which displays a start message. Choose a key to run a *start_game* function when it is pressed.

Note: *The display turtle should be defined globally as it will be needed from other functions.*

b. Clear the start display when the game begins then *listen* for the other keyboard strokes.

c. Your actual program should now set up the page and run the start function.

```
def wait_start():
    StartDisplay.speed(0)
    StartDisplay.color("re
    StartDisplay.penup()
    StartDisplay.setposit
    StartDisplay.write("Pr
    StartDisplay.hideturt

    window.onkey(start_gar
    window.listen()


def start_game():

    StartDisplay.clear()
    create_invaders()
    window.onkey(go_right,
```

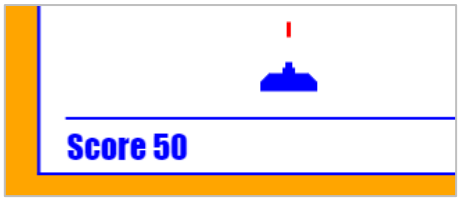

```
setup()
design_cannon()
wait_start()
```

Scoring

Create a scoring system for your game.

- The display turtle can be put in place as part of your setup function. The turtle itself and the *score* variable will need to be defined globally (i.e. outside all functions).
- Add to the score each time a space invader has been destroyed.
- Remember that if you want to concatenate text with a number (such as a score) you will need to convert the score to a string first. This can be achieved with the *str* function.
- Make any other changes that improve the display. We added a baseline.

```
global score
score += 10
score_text = "Score " + str(score)
ScoreDisplay.clear()
ScoreDisplay.write(score_text, font=
```



Lives

Decide on a number of lives the player should have in the game and display this in some way.