M30299 – Programming

Worksheet 3: Graphics

This worksheet helps you write programs that use graphics. Our aim isn't to write fully interactive graphical user interfaces (GUIs), we'll cover this later. Instead, we concentrate on using simple graphics systems that let you draw diagrams and allow basic user interaction with the mouse and keyboard.

Work through this worksheet at your own pace. As always, make sure you study each line of code and try to predict what it does. Once you have finished the instructions in this document, start working on the programming exercises and show us your solutions in the next practical session. If you need further support, book a one-to-one session with the Academic Tutors (Simon Jones or Eleni Noussi) using their Moodle page. Additionally, join the Discord channel using these instructions to ask your questions.

Importing the graphix module

The `graphix` module is not part of the standard Python libraries. You need to download it by following the instructions in Python Software (the required file is in the general section on Moodle). Make sure that the file is saved with the name `graphix.py` and is placed within a folder where Python will be able to find it (e.g., the folder where all your `.py` files are).

Now, download the `pract3.py` file and save it to the folder where your `graphix.py` and other Python files are. This file starts with the following function:

Editor

```
from graphix import Window, Point, Text

def hello_graphix():
    win = Window()
    message = Text(Point(200, 200), "Hello graphix!")
    message.draw(win)
```

Run `pract3.py` and call `hello_graphix` in the shell (as shown below). A window should appear with a hello world message.

If the window disappears straight away then (for example, in Visual Studio Code or if you've not changed these settings for PyCharm) you need to keep the created windows open by adding the statement `win.get_mouse()` to the end of the `hello_graphix` function. You can then close the window by clicking anyway on it. Alternatively, we recommend switching to Thonny.

Shell

```
hello_graphix()
```

Graphix windows

Before we write more code in `pract3.py`, let's experiment with the `graphix` module using the shell. First, let's import the `Window` class into the shell using the following code.

This line should be unnecessary if you've just executed the `pract3.py` file using Thonny since the `Window`, `Text`, and `Point` classes will remain imported. If you get an error saying there is no `graphix` module, then make sure to execute `pract3.py` — this will ensure that the Thonny shell is the correct folder to be able to locate `graphix.py`.

Shell

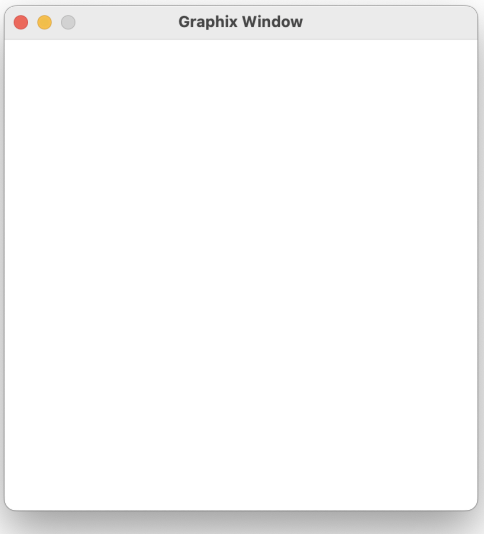
```
from graphix import Window
```

Create a `Window` object using the following assignment statement:

Shell

```
win = Window()
```

Do you see an empty window similar to the one below? You can access this window, within the shell, using the `win` variable.



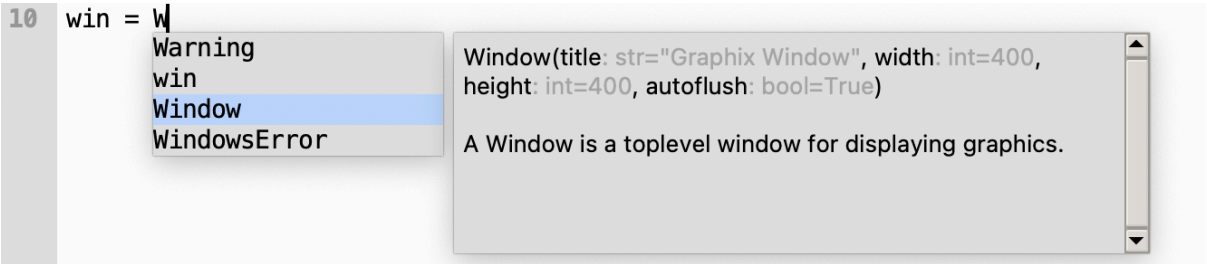
We have used the `Window` class from the `graphix` module. This is similar to how we imported and used the square root function, `sqrt`, from the `math` module, using the dot notation.

Close this window, saved in the variable `win`, by entering the following in the shell:

Shell

```
win.close()
```

Most editors come with a code completion feature, which you can activate by first typing the beginning of what you would like to use (e.g., "Win" for `Window`) and then using the shortcut `Ctrl+Space`. See the example below. The pop-up also provides a useful explanation of `Window`.



Points

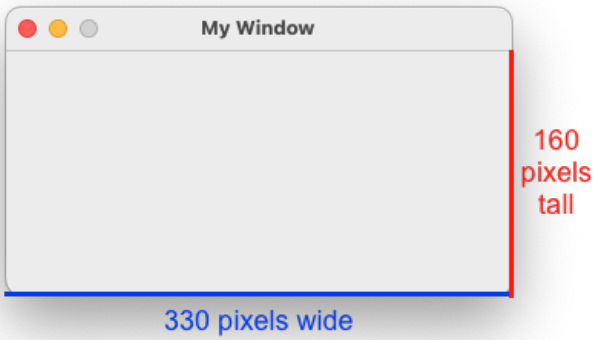
Graphix windows are 400 by 400 **pixels** by default. The top-left pixel has coordinates **(0, 0)** and the bottom-right pixel is **(399, 399)**. The x-values increase from left to right, and the y-values increase from top to bottom.

We can create windows with custom titles and sizes. Let's create a graphix window (a `Window` object) that is 330 pixels wide and 160 pixels tall, with the title "My Window". The window is saved in a variable called `my_win`:

Shell

```
my_win = Window("My Window", 330, 160)
```

You should see a window similar to the one shown below:



Next, we will create a `Point` object. Run the following line to import the Point definition:

Shell

```
from graphix import Point
```

Now we will define a `Point` object with coordinates (30, 90). To do this, enter:

Shell

```
p = Point(30, 90)
```

You will not see the point on the window. We have only created an **object** of a data type called `Point`, and assigned this object to a variable called `p`.

The `Point` data type, similar to `Window`, is defined in the `graphix` module using **object-oriented** programming. Data types that are defined using object-oriented techniques are called **classes**. You'll later learn how to define your own classes.

Objects have **attributes**, which are just data associated with an object. They also have **methods**, which are functions that let you perform actions on that object. You can use an object's attributes and method with the dot notation: Write a dot (`.`) after the object's name, and then write the name of the attribute or method.

`Point` objects have two attributes called `x` and `y` that return to us the `x` and `y` coordinates of the point. Let's see the values of the attributes of `p` (our `Point` object):

Shell

```
p.x
p.y
```

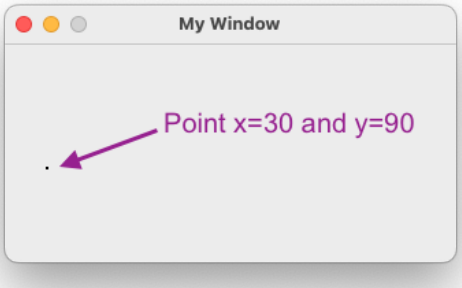
You should now see the `x` and `y` coordinates of `p`.

`Point` objects also have two methods called `draw` and `move`. Let's draw our `Point` object `p` on the window `my_win`:

Shell

```
p.draw(my_win)
```

Here we call the `draw` method of the point object `p` and pass it our `Window` object `my_win`. This should draw the point on the window as shown below:

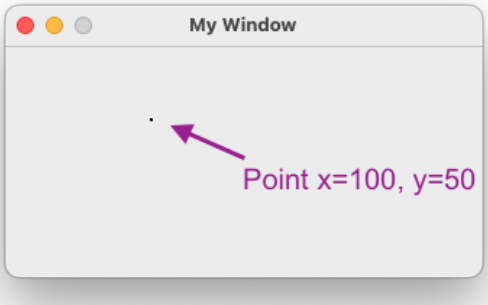


We can move this point by using the `move` method of the `Point` object. The following code moves the point 70 pixels right and 40 pixels up.

Shell

```
p.move(70, -40)
```

The Point `p` should move to a new location, as shown below. You can verify this move by printing the `x` and `y` values of `p` again.



Let's introduce another `Point` by defining a new variable `q` and drawing it on the window:

Shell

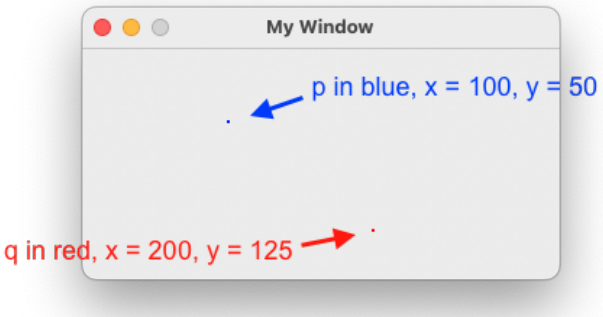
```
q = Point(200, 125)
q.draw(my_win)
```

`Point` objects, like any other graphix objects (`Circle`, `Line`, ...), also have attributes that store their colours. We can change the value of these attributes. Let's change the colours of `p` and `q` by updating the values of their `outline_colour` attributes:

Shell

```
p.outline_colour = "blue"
q.outline_colour = "red"
```

It may be hard to see them, but our points should now be painted blue and red:



Now, experiment with the attributes and the `move` method until you understand how they work. Try to move the points so that the red one (`q`) is positioned ten pixels directly above the blue one (`p`). Once you are done, close the window `my_win`. We'll make a new one for the next section.

Circles

The `graphix` module also defines types (classes) for shapes and text. Let's import the definition for circles (as well as that for `Window` and `Point`).

Shell

```
from graphix import Window, Point, Circle
```

Next, define a new window:

Shell

```
win = Window()
```

Let's first make a `Circle` object soon but first, we want the `Circle`'s centre to be at the `Point` with coordinates (200, 200):

Shell

```
centre = Point(200, 200)
```

Now we can use the `centre` variable and make a `Circle` object that has a radius of 20. The circle is stored in the variable `circle_1`:

Shell

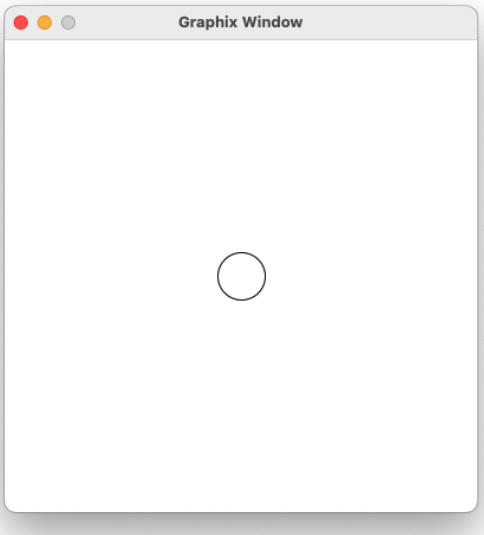
```
circle_1 = Circle(centre, 20)
```

We can then draw `circle_1` on the graphics window `win`:

Shell

```
circle_1.draw(win)
```

Do you get an output like this? Notice how the circle is placed at the centre of the window:

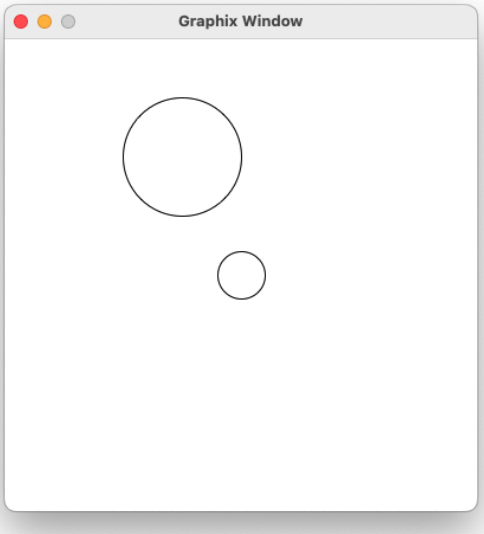


To make another `Circle`, we can create a separate `Point` variable first and use it as the centre. Or we can just put the `Point` object directly within the brackets of the `Circle`:

Shell

```
circle_2 = Circle(Point(150, 100), 50)
circle_2.draw(win)
```

This is what your output should look like:



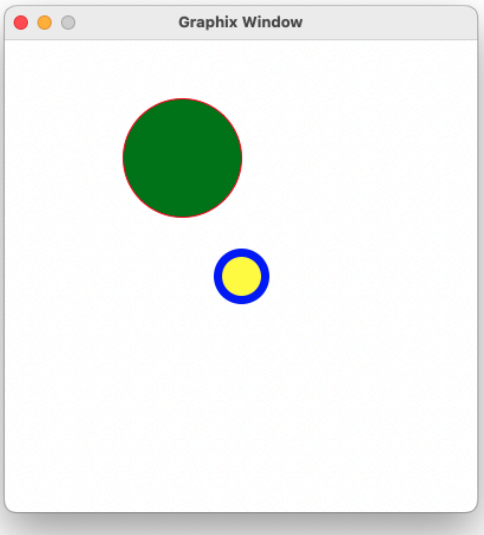
You can change shapes like `Circle`s in various ways: change the colour inside a shape by updating the value of its `fill_colour` attribute, change the colour of its border by

updating the value of its `outline_colour` attribute, and change the width of the border by updating the value of its `outline_width` attribute.

Shell

```
circle_1.outline_width = 7
circle_1.outline_colour = "blue"
circle_2.outline_colour = "red"
circle_2.fill_colour = "green"
circle_1.fill_colour = "yellow"
```

Do you get the same output as this?



`Circle` objects, just like any other graphix object, can be moved using the `move` method. We can also get copies of the radius and the centre of `Circle` objects using the `get_radius` and `get_centre` methods:

Shell

```
circle_2.move(100, 150)
print("circle_2's radius =", circle_2.radius)
print("circle_2's centre =", circle_2.get_centre())
```

Using the `move` method of the graphical objects only moves their points. In this case, we have moved the centre points of `circle_2` and the radius has remained unchanged.

Close the window before moving on to the next section.

Lines

`Line`s are specified by their endpoints (two `Point` objects). We first import the definitions:

Shell

```
from graphix import Window, Point, Line
```

Let's create a new window and draw a `Line` object:

Shell

```
win = Window()
line_1 = Line(Point(50, 150), Point(350, 250))
line_1.draw(win)
```

We can similarly move a line, set its width or change the colour of its outline:

Shell

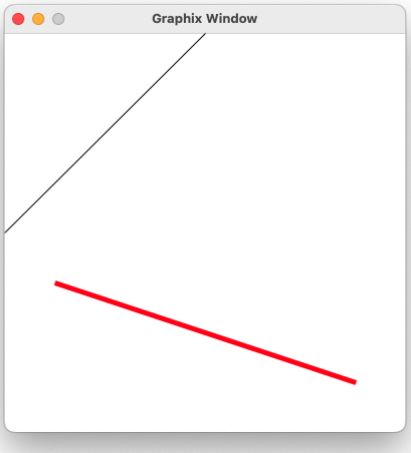
```
line_1.move(0, 100)
line_1.outline_width = 5
line_1.outline_colour = "red"
```

Below, we add another line to this window:

Shell

```
start = Point(0, 200)
end = Point(200, 0)
line_2 = Line(start, end)
line_2.draw(win)
```

This is what your two lines should look like now:



Let's print the endpoints of `line_1` after the move. Here, we are using the accessor methods `get_p1` and `get_p2` of the `Line` object to get copies of the `Point` objects at its endpoints:

Shell

```
print("line_1 starts at", line_1.get_p1())
print("line_1 ends at", line_1.get_p2())
```

Close the window before moving on.

Rectangles (and squares)

`Rectangle` objects are specified by their top-left and bottom-right `Point`s. Let's go ahead and import what we need to use them:

Shell

```
from graphix import Window, Point, Rectangle
```

In a new window, make a rectangle and set its colour to black:

Shell

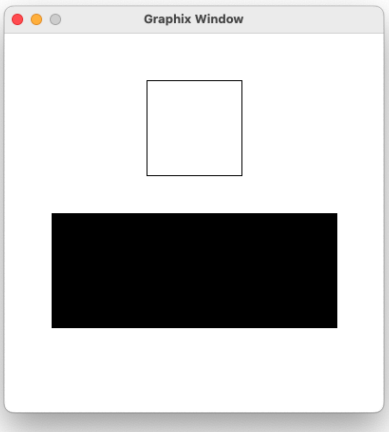
```
win = Window()
rectangle = Rectangle(Point(50, 190), Point(350, 310))
rectangle.draw(win)
rectangle.fill_colour = "black"
```

We can also draw squares using `Rectangle` objects. Run the following to see this:

Shell

```
square = Rectangle(Point(150, 50), Point(250, 150))
square.draw(win)
square.fill_colour = "white"
```

Your window should look like this now:



As an exercise, move the black rectangle up so it covers the white square. If this does not work, move the white square down to see what happens.

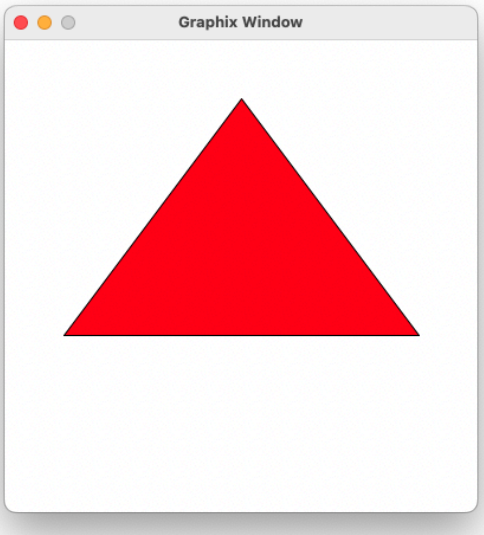
Polygons (triangles, hexagons, ...)

One way to define `Polygon` objects is by specifying a number of `Point`s for their vertices. Create a new window and draw a red triangle:

Shell

```
from graphix import Window, Point, Polygon
win = Window()
points = [Point(200, 50), Point(50, 250), Point(350, 250)]
triangle = Polygon(points)
triangle.draw(win)
triangle.fill_colour = "red"
```

You should have the same output shown below:



When defining a `Polygon`, we must pass it a **list** of `Point`s. We have not yet covered Python lists in detail (we'll see these later in the module)). All we need to know here is that the variable `points` has a value that is a list of `Point` objects (which are placed between the square brackets `[ ]`). Try this second example:

Shell

```
win = Window()
points = [
    Point(200, 50), Point(150, 100), Point(150, 200),
    Point(200, 250), Point(250, 200), Point(250, 100)
]
hexagon = Polygon(points)
hexagon.draw(win)
```

Text objects

`Text` objects are used to display text on the `graphix` window. As shown below, `Text`s are specified using a centre point and a string:

Shell

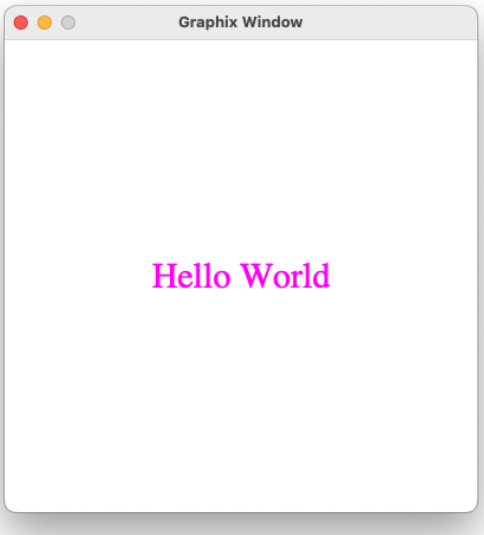
```
from graphix import Window, Point, Text
win = Window()
message = Text(Point(200, 200), "Hello World")
message.draw(win)
```

You can change the text size, the font, and the colour of the text by modifying attributes of the `Text` object:

Shell

```
message.size = 30
message.typeface = "times roman"
message.text_colour = "magenta"
```

Your text should now look like this:



We can also change the text that is being shown in a `Text` object:

Shell

```
message.text = "Goodbye"
```

Entry objects

You can use the `Entry` object to allow the user to enter strings into a `graphix` window. You can create an `Entry` on a given `Point` and with a specified width:

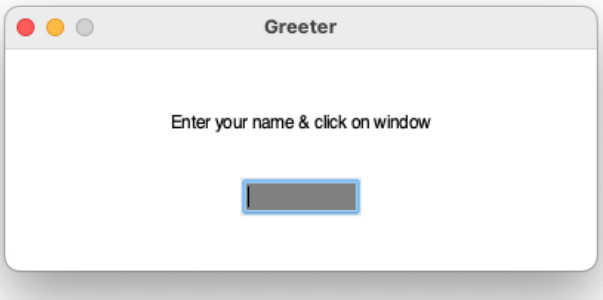
Shell

```
from graphix import Window, Point, Text, Entry
win = Window("Greeter", 400, 150)
input_box = Entry(Point(200, 100), 10)
input_box.draw(win)
```

`Entry` objects have a `text` attribute that returns the string entered by the user. Let's use it to display a user's message in the `Text` object:

Shell

```
message = Text(Point(200, 50), "Enter your name & click on window")
message.draw(win)
```



Below, we call the `get_mouse` method of the `Window` object (`win`). We will learn more about this shortly. For now, we just need to remember that it waits for the user to click on the window. We will then get the user's input from `input_box` (using the `text` attribute of `Entry`). Finally, we will display this text in `message` using the `text` attribute of the `Text` object.

Enter the following into the shell. After the first line, the window will wait for you to click on it. **Before** clicking, type in a message in the entry box (then click anywhere on the window). You will then be able to enter the final two lines of code:

Shell

```
win.get_mouse()
user_input = input_box.text
message.text = "Hello, " + user_input
```

The plus operator (+) in the final line joins two strings together.

Interactive graphics

Let's look at a simple way to make graphics programs interactive using mouse clicks.

We can obtain the position of the user's clicks on a window using the `get_mouse` method of the `Window` class. `get_mouse` method is called, and the window waits for the user to click on it. It then gives a `Point` with the coordinates of the clicked pixel.

Try the following code (you need to click on the window before entering the print statement).

Shell

```
from graphix import Window, Text
win = Window("Click Me!")
p = win.get_mouse()
print(p.x, p.y)
p.draw(win)
```

Let's use a loop to ask for 10 clicks. We also display the location of `p` using a `Text` object:

Shell

```
win = Window("Keep on clicking me!")
for i in range(10):
    p = win.get_mouse()
    x = p.x
    y = p.y
    location = Text(p, str(x) + " " + str(y))
    location.draw(win)
```

Press enter a couple of times to tell the shell that you are finished entering your code. In this example, we use the `str` function to convert the `int` returned by `x` and `y` to a string.

Programming exercises

Your solutions to this week's exercises should be added to your `pract3.py` file.

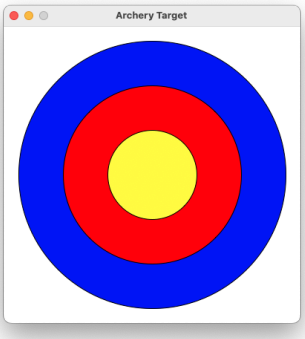
1. The `draw_stick_figure` function below is incomplete. Finish it by drawing the arms and legs.

Editor

```
def draw_stick_figure():
    win = Window()
    head = Circle(Point(200, 120), 40)
    head.draw(win)
    body = Line(Point(200, 160), Point(200, 240))
    body.draw(win)
```

2. Write a `draw_circle` function which asks the user for the radius of a circle and then draws the circle in the centre of a graphics window.

3. Write a `draw_archery_target` function that draws a target made of yellow, red, and blue circles. The sizes of the circles should be in correct proportion – i.e. the red circle should have a radius twice that of the yellow circle, and the blue circle should have a radius three times that of the yellow circle. The target should fill most of the window, so should look something like this:



4. Write a `draw_rectangle` function which asks the user for the height and width of a rectangle first. Your function should draw the rectangle in the centre of a graphics window of size 400 × 400. There should be equal spaces to the left and right, and above and below the rectangle. Assume that the user enters values less than 400.

5. Write a `blue_circle` function that allows the user to draw a blue circle of radius 100 by clicking on the location of the circle's centre.

6. The function `draw_line` in the `pract3.py` file allows the user to draw a line by choosing two points. Notice how we use the `get_mouse` method to get the `Point` from the user. Write a function `ten_lines` that allows the user to draw 10 such lines. Hint: Combine the code from `draw_line` with a loop. Also, check out the Interactive graphics section of this worksheet.

7. Write a `ten_strings` function which allows the user to plot 10 strings of their choice at locations of a graphics window picked by clicking on the mouse (the strings should be entered one-by-one by the user within a text entry box at the top of the graphics window, clicking the mouse after entering each one).

8. Write a `ten_coloured_rectangles` function to allow the user to draw 10 coloured rectangles on the screen. The user should pick the coordinates of the top-left and bottom-right corners of every rectangle by clicking on the window.

   The user needs to select the colour of each rectangle by entering a colour, for example blue, in a provided entry box at the top of the window. (The colour of each rectangle is given by the string that is in this box when the user clicks its bottom-right point.) The entry box should initially contain the string `"blue"`. Assume that the user never enters an invalid colour into the entry box.

9. [harder] Write a `five_click_stick_figure` function that allows the user to draw a (symmetric) stick figure in a graphics window using five clicks of the mouse to determine the positions of its features, as illustrated in the figure below. Each feature should be drawn as the user clicks the points.

   Hint: the radius of the head is the distance between points 1 and 2 — see the previous worksheet. Note that only the y-coordinate of point 3 should be used—its x-coordinate should be copied from point 1.