41 GUI automation

41.1 About

Python is a great tool for automating everyday tasks, such as file handling or sending emails. It can also be used for automating desktop activity such as key presses and mouse movement. The **pyautogui** library is used for this task. It can also be used to capture screenshots and other tasks.

41.2 Installing Pyautogui

Pyautogui is not installed by default and must be installed using pip. Use the following command

```
pip install pyautogui
```

Note that on Mac and Linux, there are some dependencies that need to be installed separately- refer to the Pyautogui documentation at https://pypi.org/project/PyAutoGUI/ for more information.

41.3 Create a project

41.3.1 Create the project

In PyCharm create a new project called **GUIAutomationProject**.

41.3.2 Import Pyautogui

In the project's main.py, import the library.
We'll be using some random numbers and time functions in this activity, so import those modules too.

```
import pyautogui
import random
```

```
import time
```

41.3.3 Make it failsafe

Because Pyautogui controls the mouse and keyboard, when a program is running it can be difficult to get control back if you need to stop the program, and unexpected things can happen. Pyautogui has a failsafe mode to prevent this from happening- when failsafe is enabled, moving the mouse to the top-left corner of the screen will raise an exception. The exception can be handled by the program, or it can just cause the program to crash out.
We will also add a pause- this is to add a delay between Pyautogui commands to make it clearer what is happening. The pause value is in seconds.

```
pyautogui.FAILSAFE = True
pyautogui.PAUSE = 2
```

41.4 Investigate the screen

Pyautogui can control mouse movements, using co-ordinates on the screen. Co-ordinates are expressed as **x, y** values where the **x** is the position across the screen, the **y** value is the position down the screen. The top-left corner of the screen is position **0, 0**. Note that if you have more than one monitor, Pyautogui will only work on the primary monitor.
It helps to know how big the screen is to avoid moving the mouse off the screen.
Add this line:

```
print(pyautogui.size())
```

Save and run. This will return the screen width and height:

```
Size(width=1920, height=950)
```

Save the screen size to variables by adding this line:

```
width, height = pyautogui.size()
```

41.5 Moving the mouse

The **moveTo** method is used to move the mouse to a position on the screen. The **position** method returns the current position.
Add these lines to the program:

```
pyautogui.moveTo(width / 2, height / 2)
print(pyautogui.position())
```

The script moves the mouse to the centre of the screen and reports the new position.

```
Point(x=960, y=475)
```

Note- if you are running the program in a virtual machine or on a remote desktop, you might not see the pointer move, but it should still report the correct position.

The **duration** argument of the **moveTo** command determines how long it takes to move the mouse to the new position. If it is omitted, the movement is instant.
Add some random movements to the script- use the random number generator to pick new x and y values, then use **moveTo** to move to the new position:

```
for i in range(0, 10):
    nX = random.randint(0, width - 1)
    nY = random.randint(0, height - 1)
    pyautogui.moveTo(nX, nY, duration=1)
    print(pyautogui.position())
```

Run the program. The output will look similar to this:

```
Point(x=1652, y=526)
Point(x=911, y=23)
Point(x=938, y=522)
Point(x=718, y=210)
Point(x=802, y=557)
Point(x=174, y=125)
Point(x=750, y=117)
Point(x=281, y=700)
Point(x=668, y=399)
Point(x=454, y=708)
```

Remember that we set the **PAUSE** setting for Pyautogui commands so each command will take three seconds to execute- one second to move the mouse, then a two second pause after each command.

41.6 Checkpoint

At this point your script should look like this.

```
import pyautogui
import random

pyautogui.FAILSAFE = True
pyautogui.PAUSE = 2

print(pyautogui.size())
width, height = pyautogui.size()

pyautogui.moveTo(width / 2, height / 2, duration=1)
```

```
print(pyautogui.position())

for i in range(0, 10):
    nX = random.randint(0, width - 1)
    nY = random.randint(0, height - 1)
    pyautogui.moveTo(nX, nY, duration=1)
    print(pyautogui.position())
```

41.7 Get the mouse position

The **position** command can be used to get the current mouse position when the mouse is moved by the user.
Comment out all the previous code except the imports and the PAUSE and FAILSAFE commands.
Add these lines instead:

```
while True:
    print(pyautogui.position())
    time.sleep(1)
```

Run the program and move the mouse around the screen while the program is running. The output will be similar to before:

```
Point(x=875, y=505)
Point(x=845, y=730)
Point(x=688, y=704)
Point(x=667, y=610)
Point(x=1071, y=665)
Point(x=1235, y=514)
```

You can use this program to locate items on the screen for future use. For example, open Paint on your computer, then run the program. Move the mouse to a position near the top left corner of the canvas (drawing area), wait a moment, then move it to a position near the bottom right of the canvas. The output will be similar to this (depending on your screen size):

```
Point(x=26, y=200)
Point(x=26, y=200)
Point(x=26, y=200)
Point(x=863, y=587)
Point(x=1875, y=940)
Point(x=1883, y=961)
Point(x=1884, y=960)
Point(x=1884, y=960)
Point(x=1884, y=960)
```

Although there are one or two in-between values, it is clear that the top-left position is **(26, 200)** and the bottom-right position is **(1884, 960)**.

41.8 Clicking

Pyautogui can also simulate mouse clicks. Use the click method to do this.
You can do any of the following:

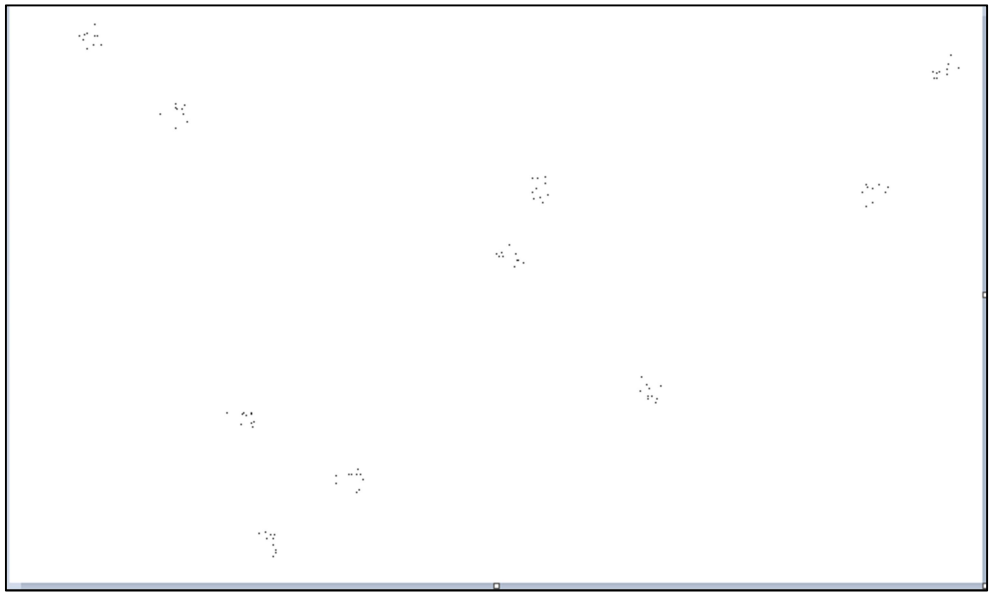
- click the mouse wherever the current pointer is- call the method with no arguments
- move to a location and click- provide **x, y** co-ordinates
- click several times- use the clicks argument, for example **clicks=5**
- choose between left-click and right-click- left is the default, or you can specify **button=”right”** or **button=”middle”**

In Paint, start a new file and select the **airbrush** brush type and the largest size.
Find the dimensions of the canvas using the method described previously.
Add this code to the script- use your own x and y dimensions:

```
for i in range(0, 10):
```

```
nX = random.randint(13, 747)
nY = random.randint(197, 620)
pyautogui.click(nX, nY)
```

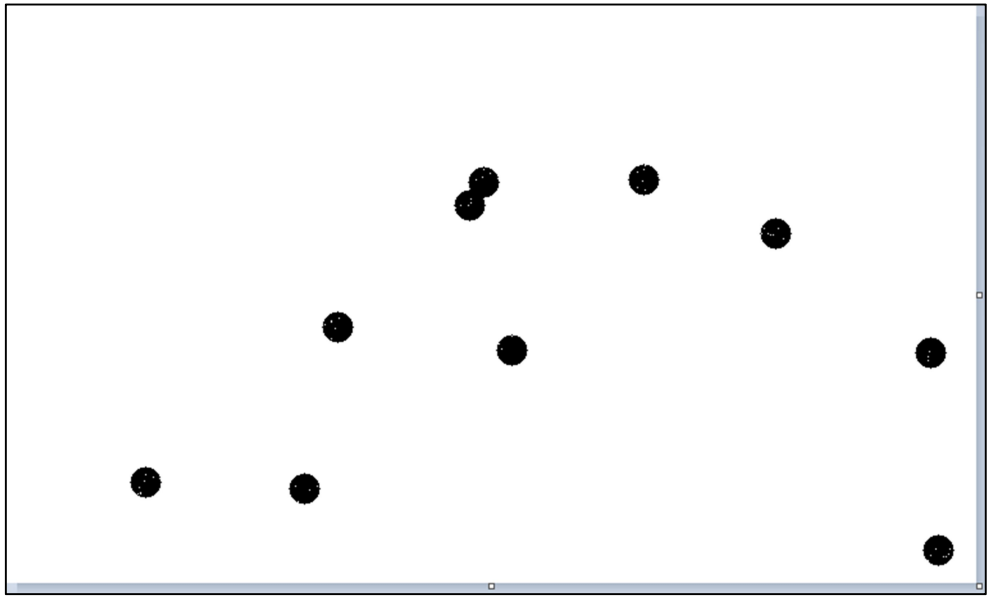
Run the script and observe the Paint window.
There will be some airbrush marks, but they will be very faint:



This shows that the clicks are instantaneous- the button is pressed down then released straight away.
Instead of clicking, sometimes it is useful to press down the button and release it separately.
Replace the click line with the following:

```
pyautogui.mouseDown(nX, nY)
time.sleep(1)
pyautogui.mouseUp()
```

Run again:



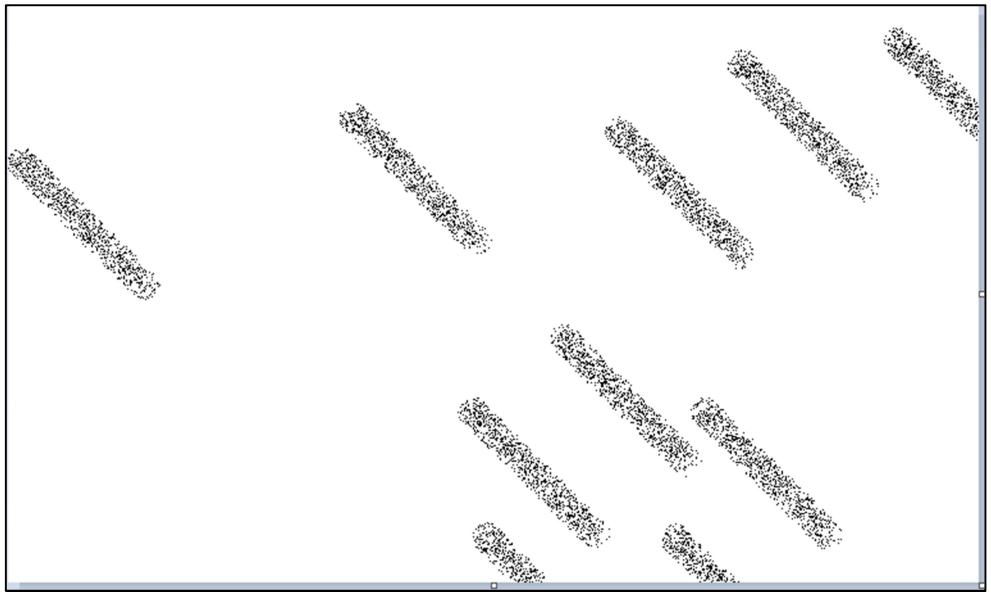
This time the airbrush marks are much more distinct, as the mouse is being held down for a second.

41.9 Dragging

It is possible to drag the mouse with Pyautogui- this involves pressing down a mouse button, moving the mouse to a new location, then releasing the mouse button. Pyautogui has a convenient **dragTo** method that does these tasks.
Change the loop code to include these lines in place of the **mouseDown** and **mouseUp** commands:

```
pyautogui.moveTo(nX, nY)
pyautogui.dragTo(nX + 100, nY + 100, 1)
```

This time when you run, the output will be similar to this:



Note that the duration of the dragTo command (third argument) determines the density of the line. Try changing it to a higher or lower values to see the effect.

41.10 Locating items

Using mouse co-ordinates to automate an application is fragile- the automation could easily be broken if the application window was in a different location on the screen. Instead, we can use image matching to locate an item (for example a button or icon) which will work no matter where on the screen the item is located.

Pyautogui is not designed to be a testing tool, but it does share some features with automated functional testing tools- image based object location is similar to how the SikuliX tool (www.sikulix.com) works.

This requires some configuration- each of the images that we want to locate must be previously captured and saved so that it can be reloaded as part of the script.

Let's try this with Paint.

41.10.1 Capture images

Open the Windows snipping tool- you can usually find this under Windows Accessories on the Windows menu. Alternatively you can use another screen capture tool such as Techsmith Capture, if you have it.

Open Paint.

Use the screen capture tool to draw around some of the shapes on the toolbar such as the star, rectangle, pentagon and triangle. Save each one with an appropriate name in your project's folder, so you have a set of images similar to this:

				
star.png	oval.png	pentagon.png	rectangle.png	triangle.png

You can choose different shapes or filenames if you wish, but remember to use the correct names in the following code.

41.10.2 Ensure the images can be located

Pyautogui has a locateOnScreen method that can be used to find an image on screen that matches the image in saved file.

In the script, comment out all the previous sections except the imports, PAUSE and FAILSAFE.

Add lines to the script as follows.

```
shapes = ["star.png", "rectangle.png", "pentagon.png", "oval.png", "triangle.png"]

for shape in shapes:
    print("{} : {}".format(shape, pyautogui.locateOnScreen(shape)))
```

Run the script. The output should be similar to this:

```
star.png : Box(left=478, top=115, width=22, height=21)
rectangle.png : Box(left=500, top=65, width=27, height=24)
pentagon.png : Box(left=478, top=91, width=22, height=24)
oval.png : Box(left=480, top=67, width=21, height=20)
triangle.png : Box(left=576, top=68, width=20, height=20)
```

Note that the locateOnScreen method reports the co-ordinates and size of the object on screen that matched the image in the file.
If the image is not found, the method returns None. For example:

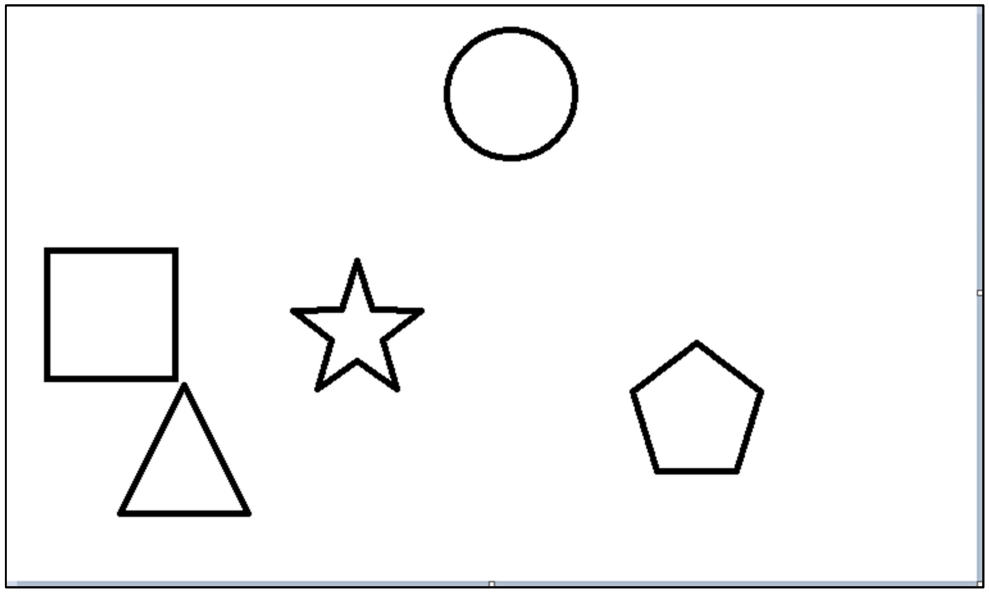
```
triangle.png : Box(left=576, top=68, width=20, height=20)
nomatch.png : None
```

41.10.3 Click the images

Images that are found on screen can be clicked using the **click** method.
Modify the program so that it clicks each image in turn, then draws a shape. We can do this by re-using some of the lines from earlier that move the mouse and drag to a different location. Add the lines as follows- use your own screen dimensions.

```
pyautogui.click(shape)

nX = random.randint(13, 600)
nY = random.randint(197, 500)
pyautogui.moveTo(nX, nY)
pyautogui.dragTo(nX + 100, nY + 100, 1)
```

The output should be similar to this.



41.11 Checkpoint

At this point the complete script should look like this.

```
import pyautogui
import random
import time

pyautogui.FAILSAFE = True
pyautogui.PAUSE = 2

shapes = ["star.png", "rectangle.png", "pentagon.png", "oval.png", "triangle.png"]

for shape in shapes:
    print("{} : {}".format(shape, pyautogui.locateOnScreen(shape)))
```

```
    pyautogui.click(shape)

    nX = random.randint(13, 600)
    nY = random.randint(197, 500)
    pyautogui.moveTo(nX, nY)
    pyautogui.dragTo(nX + 100, nY + 100, 1)
```

41.12 Keyboard commands

Pyautogui can also simulate keyboard input.
The main keyboard method is **write**, which types a given string of characters wherever the cursor is at the time.
There are also other commands-
press- used to press the special keys on the keyboard, for example the function keys, Escape, Page up/down, Caps lock
keyDown, **keyUp** - used to press and hold a key, for example a cursor key
hotkey- used to press a combination of keys such as Ctrl-C (copy) or Windows-R (opens the run dialog)

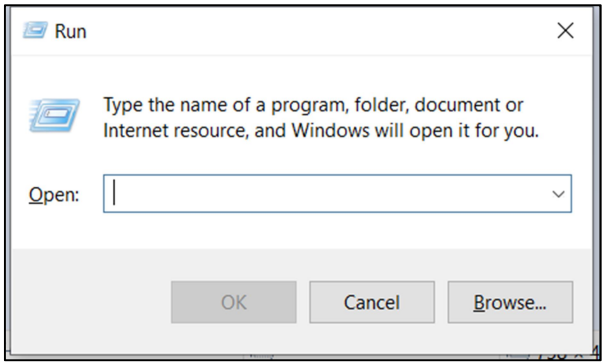
Let's test some of these with Notepad. Start by commenting-out all the code in the script except the **import**s, **PAUSE** and **FAILSAFE**.

41.12.1 Start Notepad

Start Notepad from inside the script using the Windows run command.
Add this line:

```
pyautogui.hotkey("win", "r")
```

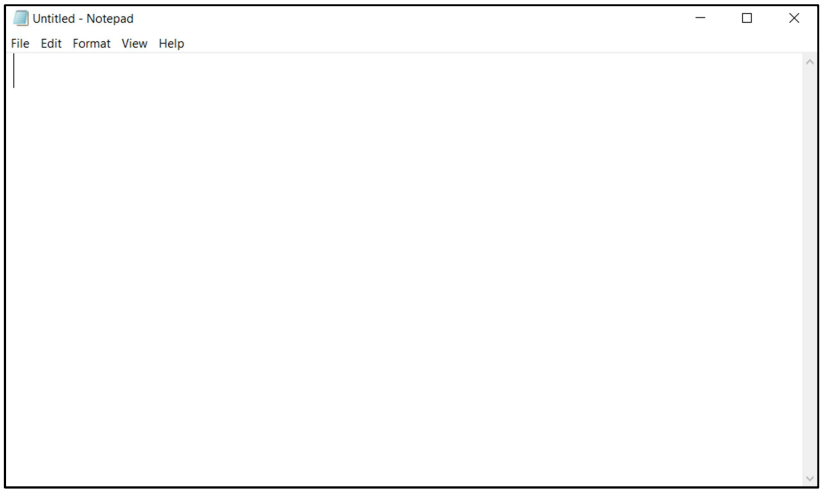
Run the script with just this line, and the Windows run dialog should appear:



Note that the focus is in the Open field. This means we can type directly into the field- we don't need to click anywhere else.
Close the box before continuing. Add lines to type in the name of the program (**notepad**) and press **Enter**.

```
pyautogui.write("notepad")
pyautogui.press("enter")
```

Run again and the Notepad window appears.



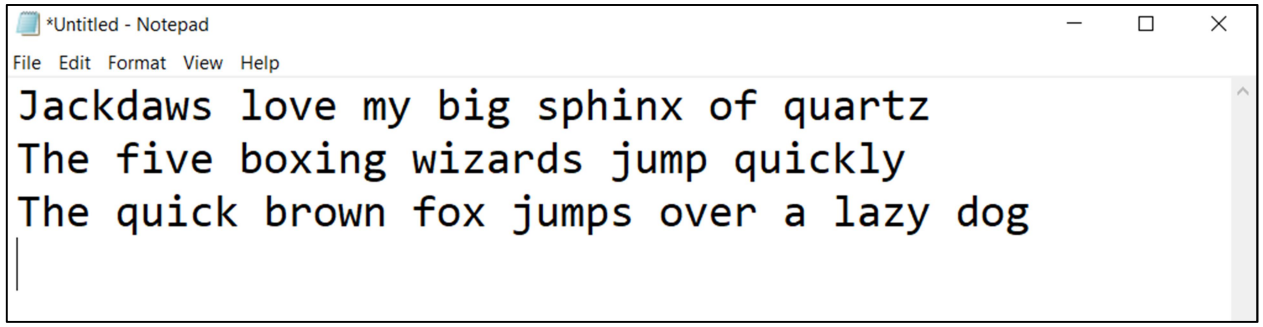
Again note that the focus is already in the main text entry window.

41.12.2 Type some text

Type some text into Notepad by adding these lines:

```
pyautogui.write("Jackdaws love my big sphinx of quartz\n")
pyautogui.write("The five boxing wizards jump quickly\n")
pyautogui.write("The quick brown fox jumps over a lazy dog\n")
```

Run the script and observe what happens.



Note that in this example the **\n** in the text is used to insert a carriage return- the same as pressing the **Enter** key.

41.12.3 Save and close Notepad

Finally let's save the file, and close Notepad.

```
pyautogui.hotkey("ctrl", "s")
pyautogui.write("pangrams.txt")
pyautogui.press("enter")

pyautogui.hotkey("alt", "f")
pyautogui.press("x")
```

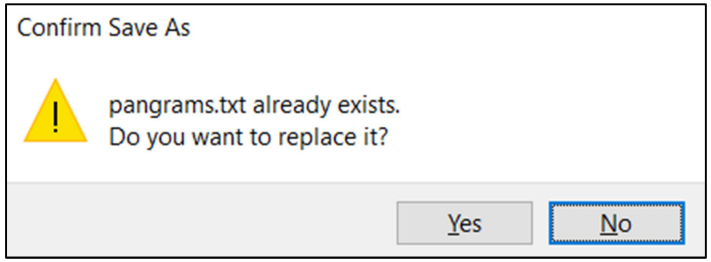
41.12.4 Run the script

Run the script and make sure the expected file is created.

Open the file to make sure it contains the correct text.

41.13 Fix a problem

Run the script again. This time, since the file already exists, a message appears at the time we want to save the file.

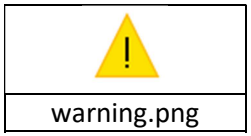


The script is blocked at this point so stop the script but leave the message box on screen.

There are a number of different ways we could deal with a problem like this.

- We could make sure the file doesn't exist before we start
- We could make sure we save the file under a new name
- We could close the dialog box and continue with the save

In this case we will use the third option. Use the snipping tool (or your screen capture tool) to capture the warning triangle image and save it in the project folder with an appropriate name.


warning.png

We can use Pyautogui's locate functions to check if the triangle exists on the screen.
Add these lines after the Enter keypress on the save dialog:

```
if pyautogui.locateOnScreen("warning.png") is not None:
    pyautogui.press("y")
```

Run the script a few times. It should work successfully and clear the dialog box each time. Check the properties of the text file to make sure it is being saved each time the script runs.

41.14 Checkpoint

At this point the script should look like this:

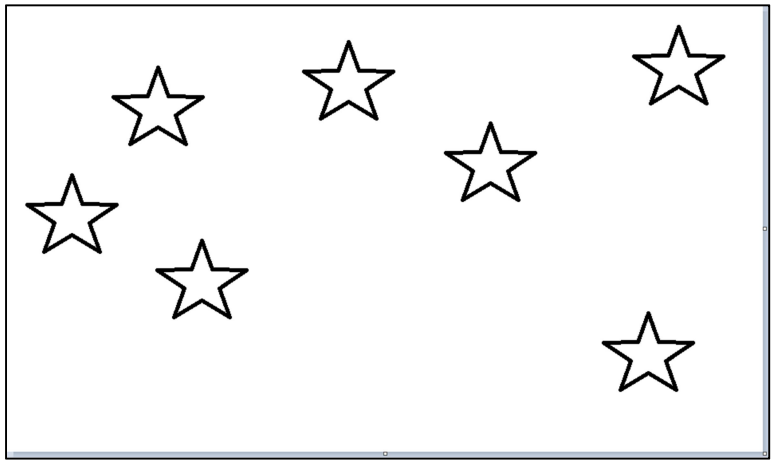
```
import pyautogui
import random
import time

pyautogui.FAILSAFE = True
pyautogui.PAUSE = 2

pyautogui.hotkey("win", "r")
pyautogui.write("notepad")
pyautogui.press("enter")

pyautogui.write("Jackdaws love my big sphinx of quartz\n")
pyautogui.write("The five boxing wizards jump quickly\n")
pyautogui.write("The quick brown fox jumps over a lazy dog\n")
```

```
pyautogui.hotkey("ctrl", "s")
pyautogui.write("pangrams.txt")
pyautogui.press("enter")

if pyautogui.locateOnScreen("warning.png") is not None:
    pyautogui.press("y")

pyautogui.hotkey("alt", "f")
pyautogui.press("x")
```

41.15 Explore more

41.15.1 Count the stars

We have seen that Pyautogui can locate an object on screen by matching it against a saved image file.
What if the image appears on the screen several times- for example like the stars in this picture?



Pyautogui has a **locateAll** method that can be used to find a set of matching objects that appear on the screen. Read about the function in the Pyautogui documentation at https://pyautogui.readthedocs.io/en/latest/screenshot.html#the-locate-functions.
In Paint, draw a picture like the one above and use the method to count how many stars appear on the screen.

41.15.2 Capture screenshots

Pyautogui has a **screenshot** method that can capture a snapshot of the whole screen, or a region of the screen. Go back to the script that draws shapes in Paint, and add the **screenshot** method after the shapes are drawn to save the file to disk.
Read about the **screenshot** method at https://pyautogui.readthedocs.io/en/latest/screenshot.html#the-screenshot-function

41.16 Note: what about web applications?

Pyautogui can be used to automate any application that appears on the desktop. That includes any normal desktop applications such as word processors, spreadsheets and even games (although it it not fast enough to automate action games). For web applications, we can also use Pyautogui, but web applications have their own problems. For example, browsers can be resized or zoomed, or objects might look a bit different in Firefox, Chrome, Edge and so on.
Instead of Pyautogui, it would be better to use an automation tool which is designed for working with web applications. That is why we use Selenium, which is covered elsewhere in this training course.

42 Challenge : Automatic Jackson Pollock

42.1 About

Jackson Pollock (1912-1956) was an American artist whose most famous works are abstract artworks made up of streaks of randomly dripped paint of different colours.

42.2 Challenge

Use Pyautogui to write a program that generates a random artwork influenced by Jackson Pollock.
‘

42.3 Getting started

Open Paint and make it full screen. Make the canvas (painting area) as big as possible.
Paint’s colour palette appears in the toolbar:

Use the mouse position script from earlier to find the x,y co-ordinates of the top-left and bottom-right of the colour palette and the painting canvas. Make a note of these positions.
In Paint, choose a suitable brush type (the oil brush type looks good) and the thickest size.

42.4 The script

Write a script that picks a random colour from the palette, then draws a line between two random points on the screen. Repeat this 200 times.

42.4.1 Hints

To pick a colour, move the mouse to anywhere in the colour palette and click.

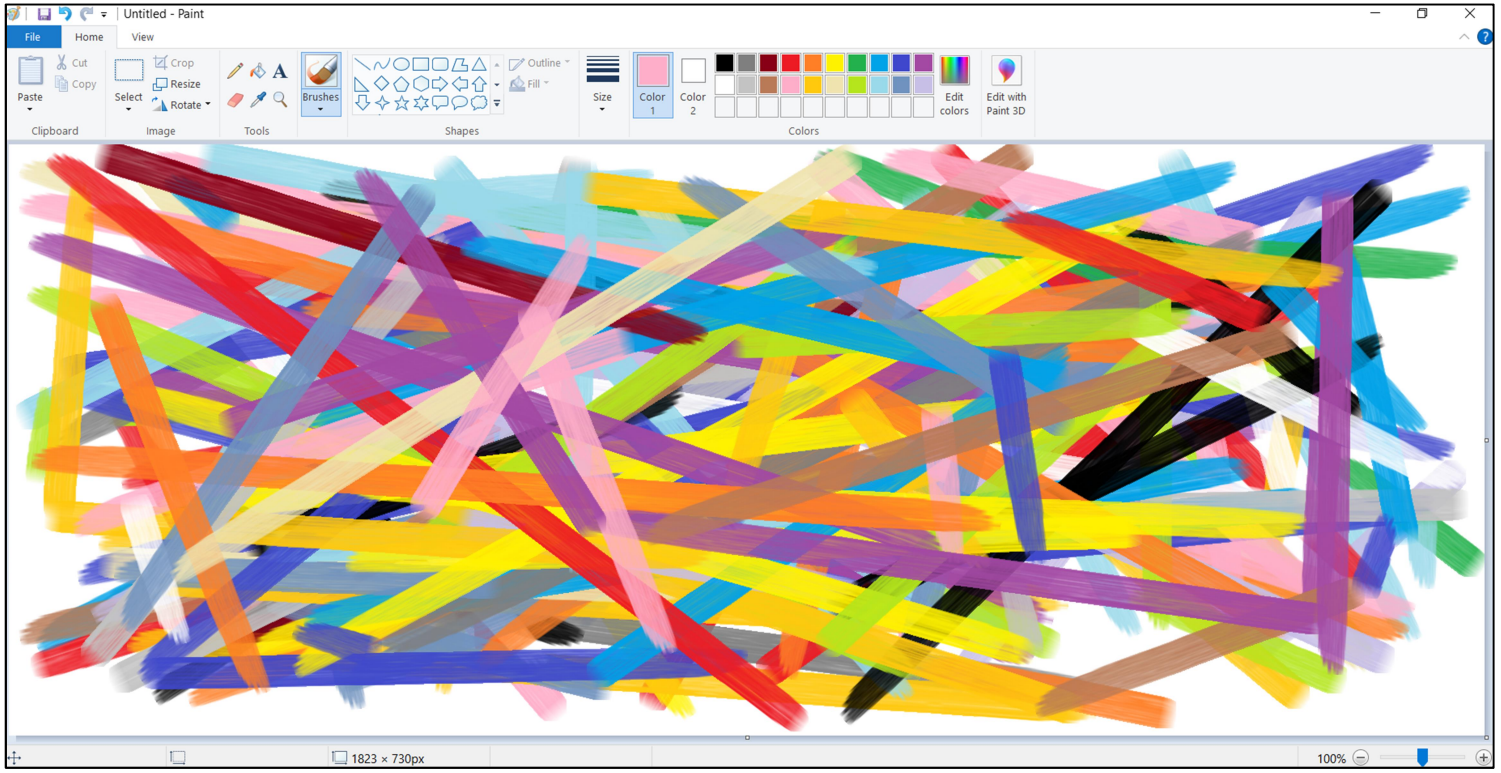
To draw a line, pick a starting point (x, y) anywhere in the canvas area. Then, drag the mouse from the starting point to the ending point (x, y)

Use **randint** to pick the various x and y values.

Because we know how many times we want to pick a colour and draw a line, use a **for** loop with a **range**.

42.5 The end result

When the program is finished, the result should look similar to this- but of course every artwork will be unique.



42.6 Improvements

In reality Pollock's work was not composed of straight lines. Actually the output from this program is more like some of the works of Kazimir Malevich (1879-1935), in particular his "suprematism" paintings, but these were composed of lines of different styles and widths.
Modify the program so that it picks a different line width after every 10 lines.

Modify the program so that it takes a screenshot when it is finished and saves it to the project folder.

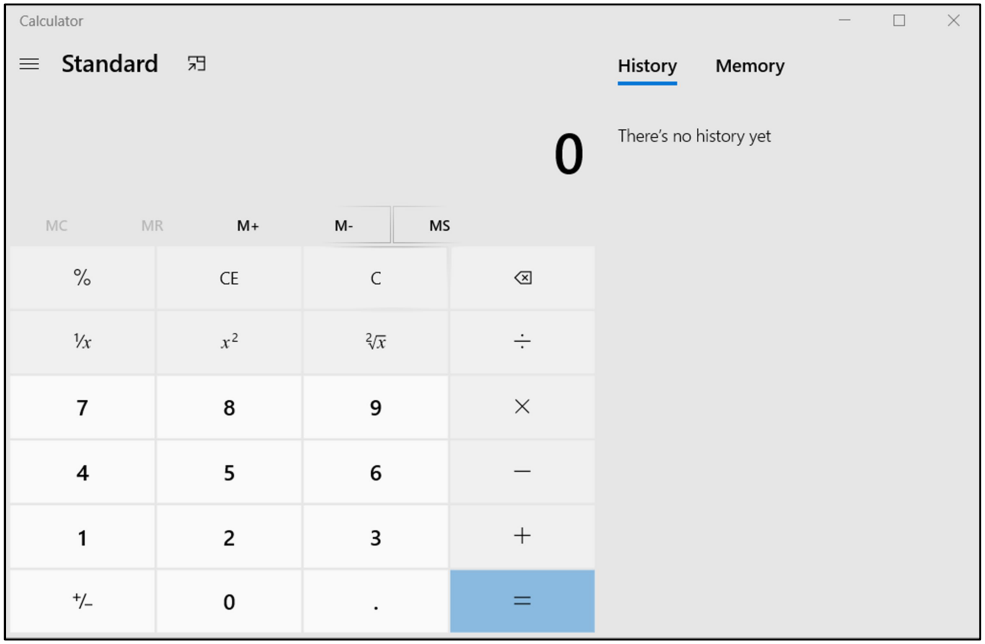
42.7 When you are finished

Print your masterpiece and sell it for $20 million.

43 Challenge : Calculator automator

43.1 About

In this challenge you will use Pyautogui to automate tasks in the Calculator program.



43.2 The challenge

43.2.1 Tasks

Use Pyautogui to do the following:

- Open Calculator
- Enter the sum **22x55**

- Press the equals button
- Capture the result as a screenshot and save it to a file
- Close Calculator

The results file should look like this:

43.2.2 Hints

Use the keyboard commands **hotkey**, **write** and **press** to open the calculator. Note that you only need to enter **calc** in the run dialog.

To enter the calculation, you have two choices:

- use the keypad on screen as if you were clicking the buttons with your mouse. You will need to capture pictures of the buttons you need and use the **locateOnScreen** method to find them to enter the calculation.
- Use the **press** method to input keypresses

Use the **screenshot** method to capture the result.

43.3 Improvements

Change the program so it does a series of calculations which are stored in a list. For example:

```
calcs = [“22*55”, “123+234”, “456/3”, “657-432”]
```

Each calculation should store a screenshot file. Use the calculation itself as the name of the file.

Hint: in Windows it is not permitted to use / or * in file names. You will need to replace those characters with something else, for example the words times or divide. This line might help:

```
pyautogui.screenshot(calc.replace("*", "times").replace("/", "divide") + ".png")
```