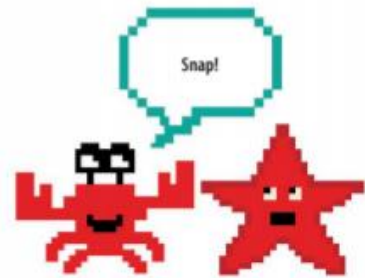


Snap

Challenge your friends to a game of digital snap. This fast-paced, two-player game requires a sharp eye and lightning-fast reactions. It works just like the card game but uses coloured shapes that appear on the screen rather than cards that are dealt.







What happens

Different shapes appear on the screen at random in either black, red, green, or blue. If a colour appears twice in succession, hit the snap key. Player 1 presses the "q" key to snap and player 2 the "p" key. Each correct snap scores a point. Snap at the wrong time and you lose a point. The player with the highest score is the winner.

▽ Starting the game

This game works in a Tkinter window. When you start the program, the Tkinter window might be hidden behind IDLE windows on your desktop. Move them out of the way so you can see the game. Be quick though: the snap shapes start appearing 3 seconds after you run the program.

<div>Snap</div> 	+	<div>Snap</div> 	=	✓	<p>This is a snap because the colours are the same, even though the shapes are different.</p>
<div>Snap</div> 	+	<div>Snap</div> 	=	X	

Although these shapes match, their colours are different, so it's not a snap.

How it works

This project uses **Tkinter** to create the shapes. **Tkinter**'s **mainloop()** function schedules a function that you'll create to show the next shape. The **random** module's **shuffle()** function makes sure the shapes always appear in a different order. The "q" and "p" keys are bound (linked) to a **snap()** function, so that each time one of these keys is pressed, it updates the relevant player's score.

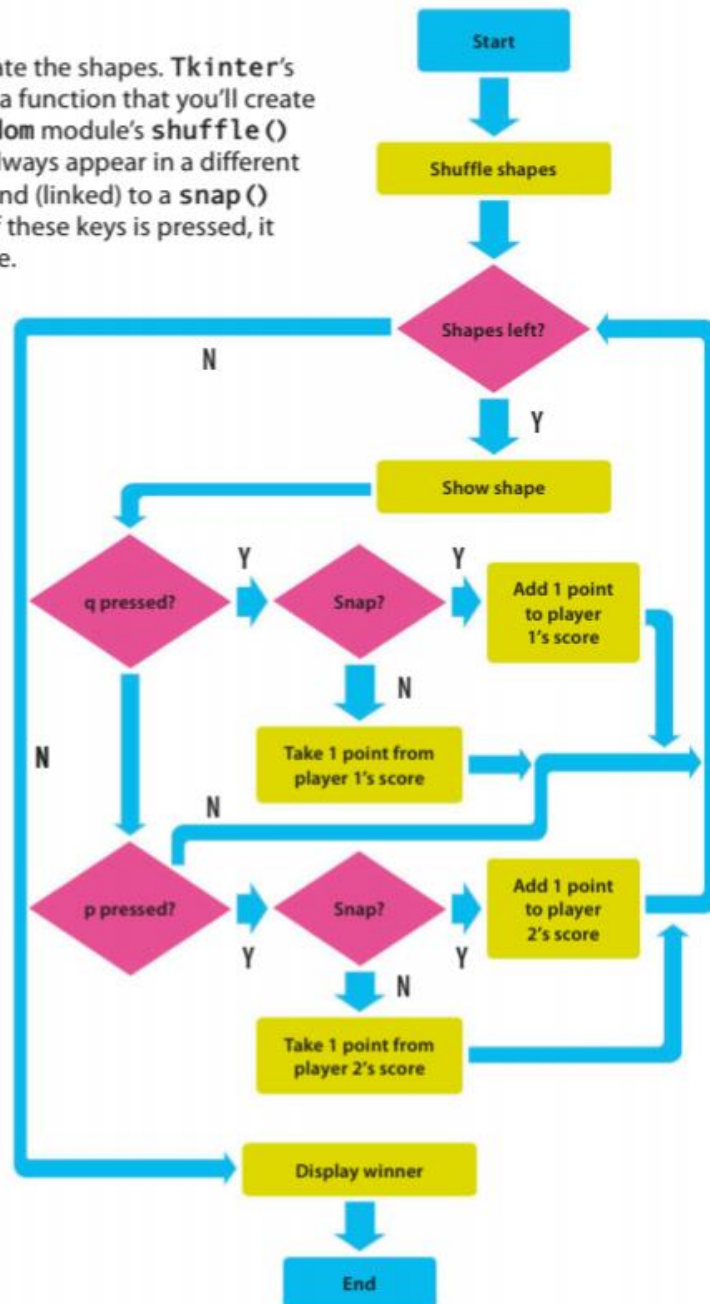
▷ Snap flowchart

The program runs for as long as there are still shapes left to be revealed. It reacts to the key presses of the players when they think they see a snap. When there are no more shapes left, the winner is declared and the game ends.

EXPERT TIPS

Sleep

Computers work a lot faster than you can. Sometimes this causes problems. If you tell a computer to show a shape to the user and then hide it again, without a break, the computer does it so quickly that the person won't see the shape. To fix this, Snap uses the time module's **sleep()** function, which pauses the program for a set number of seconds: **time.sleep(1)**, for example, puts the program to sleep for 1 second before it runs the next line of code.



Getting started

First you need to import the relevant modules and create a graphical user interface (GUI). Then you need to create a canvas to draw the shapes on.

1 Create a new file
Open IDLE. Create a new file and save it as "snap.py".

2 Add modules
First import the `random` and `time` modules, and parts of `Tkinter`. `Time` lets you create a delay so that the player is able to read a "SNAP!" or "WRONG!" message before the next shape is shown. `HIDDEN` lets you hide each shape until you want to show it with `NORMAL` – otherwise all the shapes will appear on the screen at the start of the game.

3 Set up the GUI
Now type the code shown here to create a `Tkinter` window (also called a root widget) with the title "Snap". Run the code to check it. The window may be hidden behind the other windows on the desktop.

4 Create the canvas
Type this line to create the canvas – the blank space on which the shapes will appear.



You'll shuffle the shapes using the `random` module.

```
import random
import time
from tkinter import Tk, Canvas, HIDDEN, NORMAL
```

Use `Tkinter` to create the GUI.

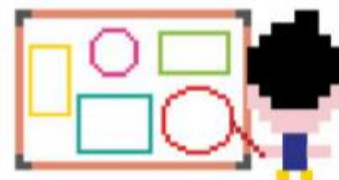
```
from tkinter import Tk, Canvas, HIDDEN, NORMAL
```

```
root = Tk()
root.title('Snap')
```

```
root.title('Snap')
c = Canvas(root, width=400, height=400)
```

Making the shapes

The next stage is to create the coloured shapes using functions from `Tkinter`'s `Canvas` widget. You'll draw circles, squares, and rectangles, each in four different colours.



5 Make a store for the shapes
You need to make a list so that you can store all the shapes somewhere. Add this line at the bottom of your file.

```
c = Canvas(root, width=400, height=400)
```

```
shapes = []
```

6

Create the circles

To draw a circle, use the Canvas widget's `create_oval()` function. Type the following code below the shapes list. It creates four circles of the same size – one each in black, red, green, and blue – and adds them to the shapes list.

Set the state to `HIDDEN` so that the shape doesn't appear on the screen when the program starts. It has to wait its turn.



Don't forget to save your work.

```
shapes = []
```

```
circle = c.create_oval(35, 20, 365, 350, outline='black', fill='black', state=HIDDEN)
```

```
shapes.append(circle)
```

```
circle = c.create_oval(35, 20, 365, 350, outline='red', fill='red', state=HIDDEN)
```

```
shapes.append(circle)
```

```
circle = c.create_oval(35, 20, 365, 350, outline='green', fill='green', state=HIDDEN)
```

```
shapes.append(circle)
```

```
circle = c.create_oval(35, 20, 365, 350, outline='blue', fill='blue', state=HIDDEN)
```

```
shapes.append(circle)
```

```
c.pack()
```

This line puts the shapes onto the canvas. Without it, none of the shapes would be displayed.

These are the (x0, y0) coordinates (see box).

These are the (x1, y1) coordinates (see box).

The circle's colour is determined by `outline` and `fill`.

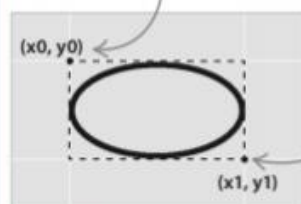


EXPERT TIPS

Create ovals

The `create_oval()` function draws an oval as if it's inside an invisible box. The four numbers within the brackets decide the position of the circles on the screen. They are the coordinates of two opposing corners of the box. The greater the difference between the two pairs of numbers, the bigger the circle.

The first pair of numbers (x0, y0) shows the position of the box's top-left corner.

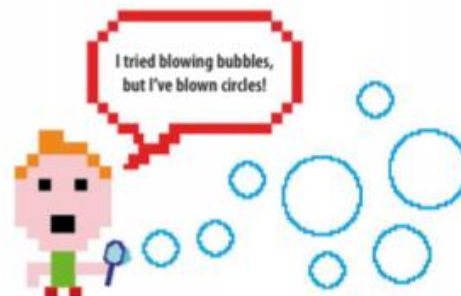


(x1, y1) shows the position of the bottom-right corner.

7

Show the circles

Try running the program. Do you see any shapes? Remember that you set their states to `HIDDEN`. Change one shape's state to `NORMAL` and run the code again. You should now be able to see that shape on the screen. Be careful not to set more than one shape to `NORMAL`. If you do, they'll all show at once, drawn one on top of the other.



8**Add some rectangles**

Now create four different-coloured rectangles using Canvas's `create_rectangle()` function. Insert this block of code between the circle-drawing code and `c.pack()`. To avoid typing it all out, just type the first two lines, then copy and paste them three times and change the colours.



Don't forget to save your work.

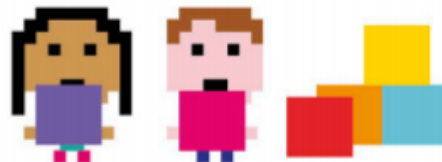
```
shapes.append(circle)

rectangle = c.create_rectangle(35, 100, 365, 270, outline='black', fill='black', state=HIDDEN)
shapes.append(rectangle)
rectangle = c.create_rectangle(35, 100, 365, 270, outline='red', fill='red', state=HIDDEN)
shapes.append(rectangle)
rectangle = c.create_rectangle(35, 100, 365, 270, outline='green', fill='green', state=HIDDEN)
shapes.append(rectangle)
rectangle = c.create_rectangle(35, 100, 365, 270, outline='blue', fill='blue', state=HIDDEN)
shapes.append(rectangle)

c.pack()
```

9**Add some squares**

Next draw the squares. You can use the same function that you used to create the rectangles, but this time you'll turn the rectangles into squares by making all their sides the same length. Add this block of code between the rectangle code and `c.pack()`.



```
shapes.append(rectangle)

square = c.create_rectangle(35, 20, 365, 350, outline='black', fill='black', state=HIDDEN)
shapes.append(square)
square = c.create_rectangle(35, 20, 365, 350, outline='red', fill='red', state=HIDDEN)
shapes.append(square)
square = c.create_rectangle(35, 20, 365, 350, outline='green', fill='green', state=HIDDEN)
shapes.append(square)
square = c.create_rectangle(35, 20, 365, 350, outline='blue', fill='blue', state=HIDDEN)
shapes.append(square)

c.pack()
```


10 Shuffle the shapes

To ensure that the shapes don't appear in the same order each time, you need to shuffle them – just like you would do with a pack of cards. The `shuffle()` function in `random` can do this for you. Insert this line after `c.pack()`.

```
random.shuffle(shapes)
```

Getting ready

In the next part of the build, you'll set up several variables and write a few bits of code that get the game ready for playing. However, it won't work until we add the functions in the last stage.



11 Set up variables

You'll need variables to keep track of various things while the program is running, including the current shape, the previous and current colour, and the two players' scores.

Neither player has any points at the start, so the value of both is set to 0.

EXPERT TIPS

Nothing really matters

Coders often need to set up variables with a starting value of zero, such as the scores in this game. But how do you do this if a variable holds a string rather than a number? The answer is to use a pair of quote marks with nothing between them. Some variables, however, don't have an obvious default value such as 0 or an empty string. In that case, you can use the word "None", as we do below.

```
random.shuffle(shapes)
```

The `shape` variable has no value yet.

```
shape = None
```

```
previous_color = ''
```

```
current_color = ''
```

The color variables hold an empty string.

```
player1_score = 0
```

```
player2_score = 0
```

12 Add a delay

Now add a line to create a 3-second delay before the first shape appears. This gives the player time to find the Tkinter window in case it's hidden behind other windows on the desktop. You'll create the `next_shape()` function later, in Steps 16 and 17.

```
player2_score = 0
```

```
root.after(3000, next_shape)
```

The program waits for 3,000 milliseconds, or 3 seconds before showing the next shape.

13 React to snaps

Next add these two lines to your code. The `bind()` function tells the GUI to listen for the "q" or "p" key being pressed, and to call the `snap()` function each time it happens. You'll create the `snap()` function later.

```
root.after(3000, next_shape)
c.bind('q', snap)
c.bind('p', snap)
```

14 Send key presses to the GUI

The `focus_set()` function tells the key presses to go to the canvas. The GUI wouldn't react to "q" and "p" being pressed without this function being called. Type this line below the `bind()` function calls.

```
c.bind('q', snap)
c.bind('p', snap)
c.focus_set()
```

15 Start the main loop

Add this line right at the end of your file. Once we add the `next_shape()` and `snap()` functions, the main loop will update the GUI with the next shape and listen for key presses.

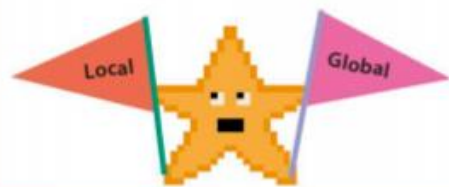
```
c.focus_set()

root.mainloop()
```

EXPERT TIPS

Local and global variables

Variables can either be local or global. A local variable exists only inside a particular function, which means the rest of the program can't use it. A variable created in the main program, outside of a function, is called global and can be used in any part of the code. However, if you want to use a function to assign a new value to a global variable, you need to add the keyword `global` before the variable's name when you type it in the function. This is what we do in Step 16.



Coding the functions

The last stage is to create two functions: one to show the next shape, and another to handle snaps. Type them at the top of your program, just below the import statements.

16 Create the function

The `next_shape()` function shows the coloured shapes one after another, like cards being dealt. Start defining the function by typing the code below. It labels some of your variables as `global` (see box, left) and updates `previous_color`.

Using the `global` keyword ensures that changes to the variables are seen throughout the program.

```
def next_shape():
    global shape
    global previous_color
    global current_color

    This line sets
    previous_color
    to current_color
    before the code
    gets the next shape.

    previous_color = current_color
```

17 Complete the function

Now type out the rest of the function. To show a new shape, we need to change its state from **HIDDEN** to **NORMAL**. The code below does this by using Canvas's `itemconfigure()` function. It uses another Canvas function, `itemcget()`, to update the `current_color` variable, which will be used to check for a snap.

```
previous_color = current_color

c.delete(shape)
if len(shapes) > 0:
    shape = shapes.pop()
    c.itemconfigure(shape, state=NORMAL)
    current_color = c.itemcget(shape, 'fill')
    root.after(1000, next_shape)
else:
    c.unbind('q')
    c.unbind('p')
    if player1_score > player2_score:
        c.create_text(200, 200, text='Winner: Player 1')
    elif player2_score > player1_score:
        c.create_text(200, 200, text='Winner: Player 2')
    else:
        c.create_text(200, 200, text='Draw')
c.pack()
```

Delete the current shape, so that the next shape doesn't show on top of it and so that it won't be shown again.

Get the next shape if there are any shapes left.

Make the new shape visible.

Assign `current_color` to the colour of the new shape.

Wait 1 second before showing the next shape.

These lines stop the program responding to snaps after the game is over.

This code shows the winner on the screen or declares the game a draw.

EXPERT TIPS

Configuring Canvas items

You can alter things that appear on the canvas by using Canvas's `itemconfigure()` function. In this game, for instance, you use `itemconfigure()` to change shapes from hidden to visible, but you could also use it to change their colour or other characteristics. To use `itemconfigure()`, put the name of the item you want to change in brackets, followed by a comma and then the characteristic and its new value.

The characteristic being changed

The name of the Canvas item you want to change.

The new value

```
c.itemconfigure(shape, state=NORMAL)
```


18

Is it a snap?

To complete the game, create your last function: `snap()`. This function will check which player has hit their key and whether the snap is valid (correct). It will then update the scores and show a message. Add this code beneath the `next_shape()` function.



Don't forget to save your work.

```
def snap(event):
    global shape
    global player1_score
    global player2_score
    valid = False

    c.delete(shape)

    if previous_color == current_color:
        valid = True

    if valid:
        if event.char == 'q':
            player1_score = player1_score + 1
        else:
            player2_score = player2_score + 1
        shape = c.create_text(200, 200, text='SNAP! You score 1 point!')
    else:
        if event.char == 'q':
            player1_score = player1_score - 1
        else:
            player2_score = player2_score - 1
        shape = c.create_text(200, 200, text='WRONG! You lose 1 point!')

    c.pack()
    root.update_idletasks()
    time.sleep(1)
```

Label these variables as global so the function can change them.

Check if it's a valid snap (if the colour of the previous shape matches the colour of the current shape).

If the snap is valid, check which player snapped and add 1 to their score.

This line shows a message when a player makes a valid snap.

Otherwise (else), take away one point from the player that snapped.

This line shows a message when a player snaps at the wrong time.

This line forces the program to update the GUI with the snap message immediately.

Wait 1 second while players read the message.

19

Test your code

Now run the program to check it works. Remember you need to click on the **Tkinter** window before it will respond to the "q" and "p" keys.

Wait 1 second while players read the message.

Hacks and tweaks

Tkinter can show lots of different colours and shapes besides circles, squares, and rectangles, so there's plenty of scope to customize your game. Here are some ideas to try out – including making the game cheat-proof!

▽ Speed up the game

You can make the game a bit harder by reducing the time delay between each shape as the game progresses. Hint: try storing the time in a variable, starting at 1000 and subtracting 25 from it each time a shape is shown. These numbers are just suggestions – experiment with them to see what you think works best.



△ Coloured outlines

The program looks at the `fill` parameter, not the `outline`, when it's judging whether a valid snap has been made. You can give different-coloured outlines to shapes and they will still make a snap so long as their fill colours match.



△ Add more colours

You may have noticed that Snap is quite a short game. To make it longer, add extra squares, rectangles, and circles using different colours.

Make new shapes

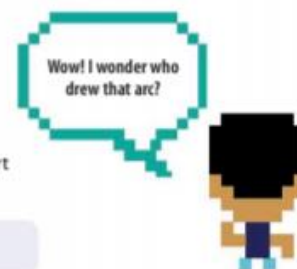
You can change the parameters of `create_oval()` to produce an oval rather than a circle. Tkinter can also draw arcs, lines, and polygons. Try out the examples shown here, and play around with the parameters. Remember to keep the `state` as `HIDDEN` to hide the shape until it's time to show it.

1 Draw arcs

Use the `create_arc()` function to draw arcs. A solid arc is drawn unless you give your arc a style. To use Tkinter's different arc styles, import `CHORD` and `ARC` by changing the third line of your program, as shown below. Then add some chords and arcs to your list of shapes, as shown overleaf.

Type this to import the arc styles.

```
from tkinter import Tk, Canvas, HIDDEN, NORMAL, CHORD, ARC
```



```
arc = c.create_arc(-235, 120, 365, 370, outline='black', \
                  fill='black', state=HIDDEN)
```



```
arc = c.create_arc(-235, 120, 365, 370, outline='red', \
                  fill='red', state=HIDDEN, style=CHORD)
```



```
arc = c.create_arc(-235, 120, 365, 370, outline='green', \
                  fill='green', state=HIDDEN, style=ARC)
```



2 Draw lines

Now try adding some lines to your list of shapes using the `create_line()` function.

```
line = c.create_line(35, 200, 365, 200, fill='blue', state=HIDDEN)
```

```
line = c.create_line(35, 20, 365, 350, fill='black', state=HIDDEN)
```



3 Draw polygons

Next try making some polygons for your shape collection, using `create_polygon()`. You'll need to give coordinates for each corner of your polygons.

The three pairs of numbers in the code give the coordinates of the triangle's corners.

```
polygon = c.create_polygon(35, 200, 365, 200, 200, 35, \
                          outline='blue', fill='blue', state=HIDDEN)
```



Stop players cheating

Right now, if a snap is valid and both players hit their snap keys at the same time, they each get a point. In fact, they will still be able to score points up until the next shape is shown, since the previous and current colour will still be the same. Try this hack to stop the players cheating.

1 Go global

First you need to say that `previous_color` is a global variable in the `snap()` function, because you need to change its value. Add this line under the other global variables.

```
global previous_color
```

2

Block a multiple snap

Next add the following line to the `snap()` function to set the value of `previous_color` to the empty string ('') after a correct snap. Now if a player presses their key again before the next shape is shown, they will lose a point. This is because '' will never be equal to the current colour, except before the first shape is shown.



There's nothing I don't know about multiple snaps!

```
shape = c.create_text(200, 200, text='SNAP! You scored 1 point!')
previous_color = ''
```

3

Prevent early snaps

Since `previous_color` and `current_color` are equal at the beginning of the game, players can still cheat by pressing their key before the first shape appears. To solve this, set the two variables to different strings at the start. Change their values to "a" and "b".

```
previous_color = 'a'
current_color = 'b'
```

Starting with different strings means that a snap can't be made until the shapes appear on the screen.

4

Change the messages

If both players press their keys at almost the same time, it might be confusing as to who has scored or lost a point. To fix this, you can change the messages that are displayed when players attempt a snap.



Don't forget to save your work.

```
if valid:
    if event.char == 'q':
        player1_score = player1_score + 1
        shape = c.create_text(200, 200, text='SNAP! Player 1 scores 1 point!')
    else:
        player2_score = player2_score + 1
        shape = c.create_text(200, 200, text='SNAP! Player 2 scores 1 point!')
    previous_color = ''
else:
    if event.char == 'q':
        player1_score = player1_score - 1
        shape = c.create_text(200, 200, text='WRONG! Player 1 loses 1 point!')
    else:
        player2_score = player2_score - 1
        shape = c.create_text(200, 200, text='WRONG! Player 2 loses 1 point!')
```