28 Recursive functions

Python supports recursion. A recursive function is a function that calls itself. Clearly, this is something that must be done with care to avoid ending up in an infinite loop, but it can be very useful in certain cases.

28.1 Before you start

Create a new project called RecursionProject or similar.

28.2 Create a basic recursive function

Add this code to create a recursive function.

```
def rf(n):
    print(f"hello {n}")
    rf(n - 1)
```

Note that when the function is originally called, a value is passed to the function. The function prints a message, then calls itself with the original value -1. Add a line to call the function:

```
rf(99)
```

Run the program. The output will show the message with the number counting down:

```
hello 99
hello 98
hello 97
hello 96
hello 95
```

But, since there is nothing in the function that stops the countdown, it is an infinite loop. Python has a recursion limit to stop this happening:

```
  File "C:\python-workspace\workbook - recursionProject\MostBasicRecursion.py", line 2, in rf
    print(f"hello {n}")
RecursionError: maximum recursion depth exceeded while calling a Python object
```

The recursion limit can be changed - see the Explore more section.

Change the function so that it has a breakout clause that prevents the infinite loop happening:

```
def rf(n):
    if n > 0:
        print(f"hello {n}")
        rf(n - 1)
    else:
        print("The end")
```

Run the program again. This time it ends nicely:

```
hello 4
hello 3
hello 2
hello 1
The end
```

28.3 Write a factorial function

A classic example of a recursive function is a function to calculate a factorial (a number multiplied by every number less than itself).
Add a new function to the program, and a line to call it. Run it and check the output is as expected.

```
def fctl(n):
    if n > 1:
        return n * fctl(n - 1)
    else:
        return n

print(fctl(8))
```

```
40320
```

Note that there is an easier way to do this- there is a **factorial** function to do this in the **math** library.

28.4 Recursive squares

Python's turtle is a good way of illustrating recursion.

Start by adding a line to import the turtle.

```
from turtle import *
```

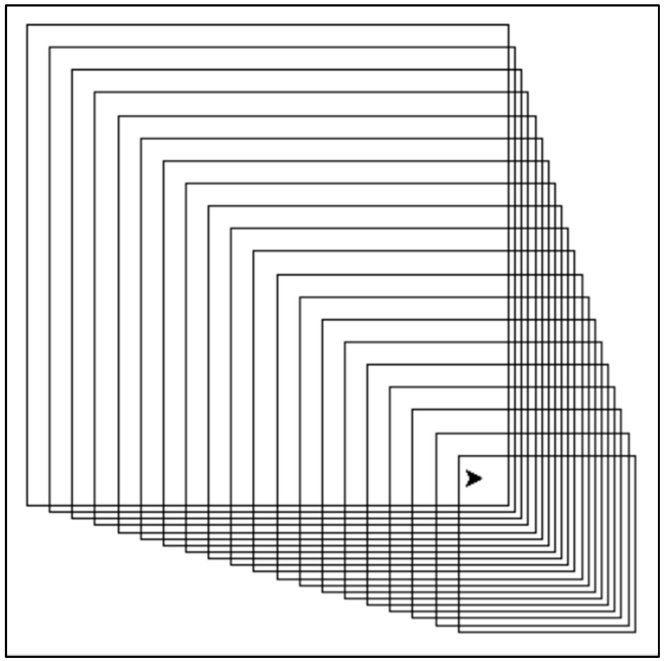
Add a function to draw a square recursively. Note that the function draws a square, then calls itself to draw a smaller square, until the smallest square is reached.

```
def recursivesquare(n):
    if n > 0:
        for i in range(4):
            forward(100 + n * 10)
            right(90)

        penup()
        right(45)
        forward(20)
        left(45)
        pendown()

        recursivesquare(n - 1)
```

Add code to reset the turtle and call the recursive square function with an initial value.

```
reset()
speed(0)

recursivesquare(20)
exitonclick()
```

Run the program. A value of **20** produces an output that looks like this.



28.5 Turtle trees

28.5.1 Draw a tree

The turtle can be used to draw some interesting natural-looking shapes.

Add a function that draw a tree of a certain size. Note that the tree function calls itself twice, to draw a symmetrical shape.

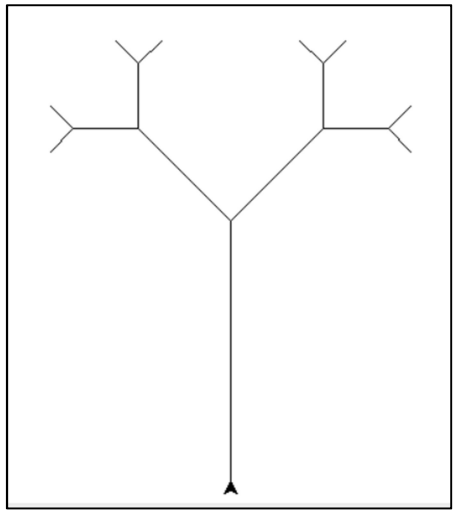
```
def tree(n):
    if n > 20:
        forward(n)
        left(45)
        tree(n * 0.5)
        right(90)
```

```
        tree(n * 0.5)
        left(45)
        back(n)
```

Call the function as follows. Note that the turtle pointer is positioned at the bottom of the screen before drawing starts. You might need to experiment a bit with the position based on your screen resolution.

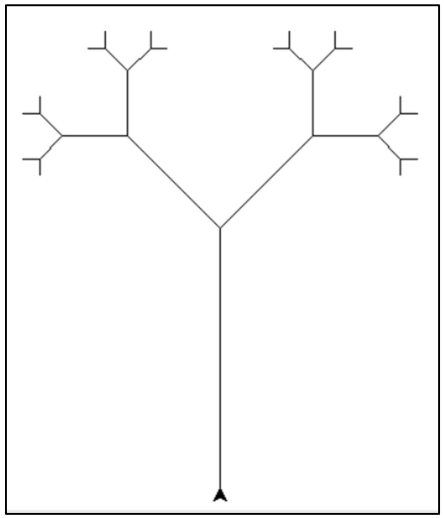
```
reset()
speed(0)
penup()
goto(0, -300)
left(90)
pendown()

tree(200)
exitonclick()
```

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

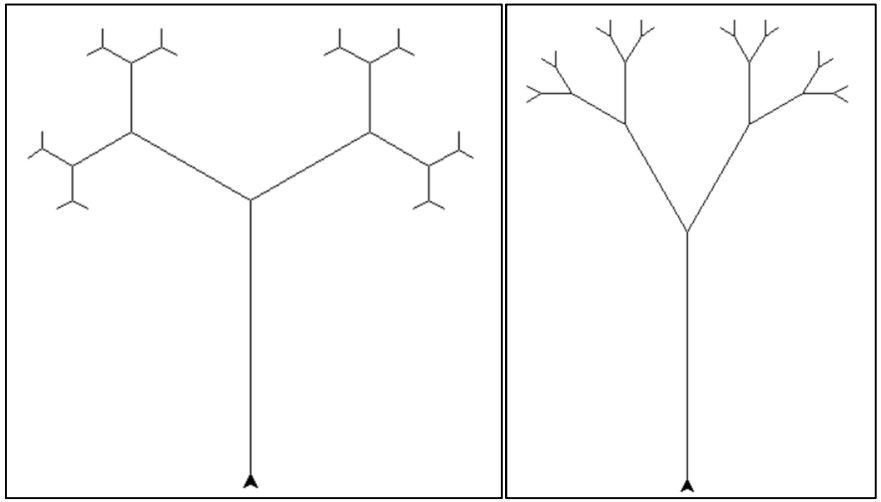


28.5.2 Tweak the program

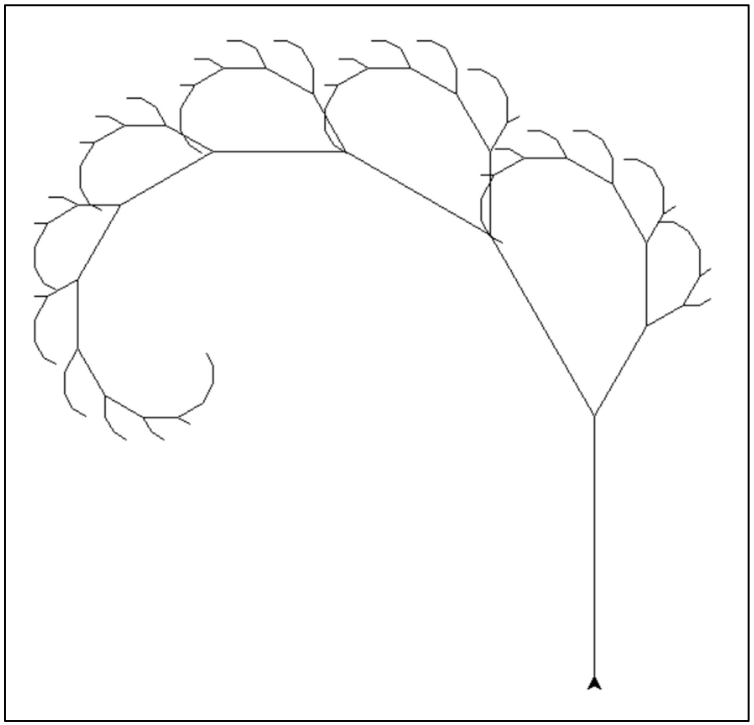
Changing the numbers in the program changes the appearance of the output. For example, changing the if statement to if n > 10: causes more branches to be drawn.



Changing the angles draws a tree that is taller and narrower, or shorter and wider. In these examples, in the screenshot on the left the left angles are 60 and the right angle is 120. In the screenshot on the right the left angles are 30 and the right angle is 60.



Changing the values in the recursive calls to the tree function also changes the look of the drawing. Try changing the first call to tree(n * 0.8) and the second call to tree(n* 0.4) to give the output a more fern-like appearance:

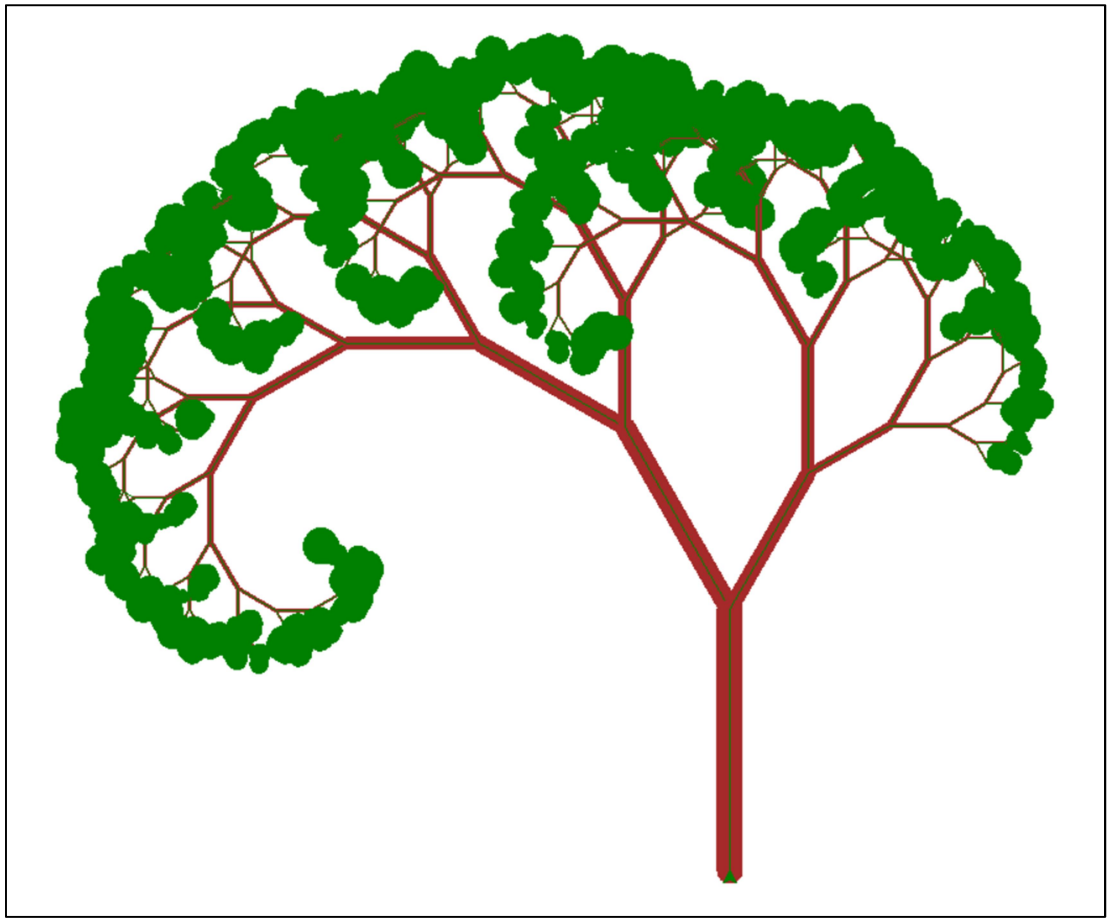


28.6 Challenges

Modify the program as follows:

- Change the width of the line so that the trunk of the tree is thicker and the branches are narrower towards the top. The turtle **width** function can be used to do this.
- Add some foliage. At the end of each branch, draw a shape such as a circle to give the impression of leaves on the tree. To make it look a little bit more natural, change the radius of the circle to a random value.
- Change the colours so that the branches are brown and the leaves are green.

The end result should look similar to this:



28.7 Explore more

28.7.1 Recursion limit

By default, Python's recursion limit is 1000.

To check the current limit, try this code.

```
import sys
print(sys.getrecursionlimit())
```

The recursion limit can be changed (with care, of course). Try changing the limit, then displaying the limit again:

```
sys.setrecursionlimit(2000)
print(sys.getrecursionlimit())
```

28.7.2 Explore more recursion

For other examples of recursion see https://en.wikipedia.org/wiki/Recursion_(computer_science)