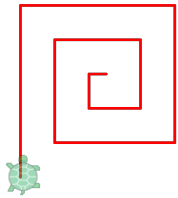
14. Recursion

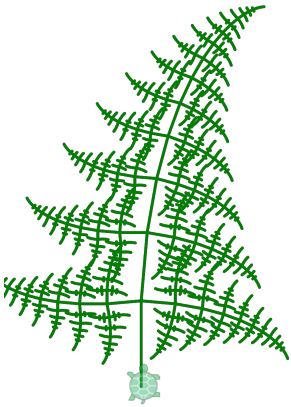
Recursive Spiral

```
spiral = (x) ->
  if x > 0
    fd x * 10
    rt 90
    spiral x - 1
    lt 90
    bk x * 10
pen red
spiral 10
```



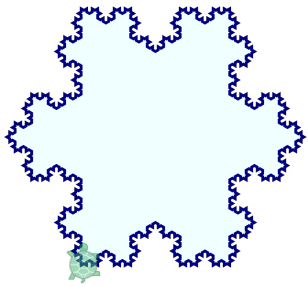
Fractal Fern

```
speed 1000
fern = (x) ->
  if x > 1
    fd x
    rt 95
    fern x * .4
    lt 190
    fern x * .4
    rt 100
    fern x * .8
    lt 5
    bk x
pen green
fern 50
```



Koch Snowflake

```
speed Infinity
flake = (x) ->
  if x < 3 then fd x
  else
    flake x / 3
    lt 60
    flake x / 3
    rt 120
    flake x / 3
    lt 60
    flake x / 3
pen 'path'
for s in [1..3]
  flake 150
  rt 120
fill 'azure strokeStyle navy'
```



Recursive functions refer to themselves, and they can achieve powerful effects. Recursion is at the core of fractals, language, and reasoning.

Recursion as a Stack

Operationally, recursion works by stepping through a stack of work. Consider the sequence as **Spiral** draws a shape and retraces it back.

```
spiral 10 sets x to 10
  rt 90; fd x * 10; spiral x - 1 ⇓                    lt 90; bk x * 10 ⇑
```

```
  spiral 9 sets x to 9
    rt 90; fd x * 10; spiral x - 1 ⇓                lt 90; bk x * 10 ⇑
```

```
    spiral 8 sets x to 8
      rt 90; fd x * 10; spiral x - 1 ⇓            lt 90; bk x * 10 ⇑
```

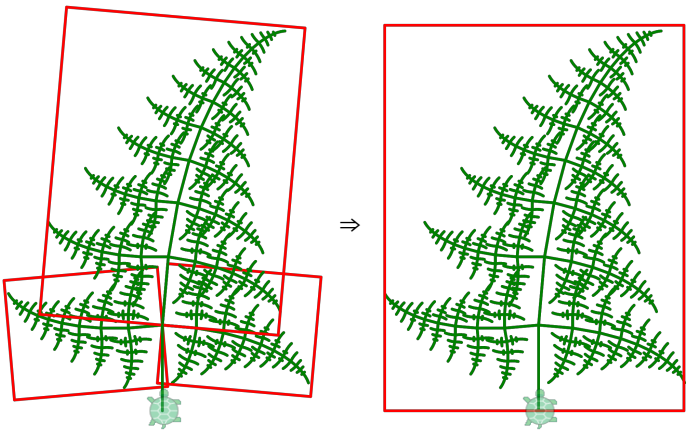
```
                ... etc, until the base case  spiral 0 ⇑
```

Each time `spiral` is called, it puts the previous call on hold and does the smaller spiral. After the smaller spiral is done, it returns to finish work on the bigger one. `spiral 0` does nothing: that is called the ***base case***.

The `x` at different levels are ***local*** variables that do not interfere with each other. Each red box is a ***stack frame*** with its own "copy" of `x`.

Recursion as a Reduction

Conceptually, recursion reduces a problem to smaller cases. Consider how **Fern** draws a large fern by assuming it can draw smaller ferns:



All `fern` does is draw a stem with three smaller ferns at the end. The main caveat is that the reduction has a limit: it ends when $x \leq 1$.

Both **Spiral** and **Fern** return the turtle to exactly the same position and direction at the end of a function call. Maintaining an ***invariant*** like this can make recursion much easier to understand.