Factorial is an example of a mathematical function that is defined recursively, i.e., it is partly defined in terms of itself.

\[ n! = \begin{cases} 
1 & \text{if } n \leq 1 \\
(n \times (n-1))! & \text{otherwise} 
\end{cases} \]

Factorial, Recursively

But we can use the recursive definition directly to get a different version

```c
/* Compute n factorial – the product of the first n integers, 1 * 2 * 3 * 4 ... * n */
int factorial(int n)
{
    int result;
    if (n <= 1) 
        result = 1;
    else 
        result = n * factorial(n - 1);
    return result;
}
```

Factorial Revisited

We've already seen an implementation of factorial using a loop

```c
int factorial(int n)
{
    int result;
    if (n <= 1) 
        result = 1;
    else 
        result = n * factorial(n - 1);
    return result;
}
```

Trace

```c
factorial(4) =
4 * factorial(3) =
4 * 3 * factorial(2) =
4 * 3 * 2 * factorial(1) =
4 * 3 * 2 * 1 =
4 * 3 * 2 * 1 =
4 * 3 * 2 * 1 =
```
What is Recursion?
Definition: A function is recursive if it calls itself

```c
int foo(int x) {
  ...
  y = foo(...);
  ...
}
```

How can this possibly work???

Function Calls
Answer: there’s nothing new here!
Remember the steps for executing a function call in C:
Allocate space for called function’s parameters and local variables
Initialize parameters
Begin function execution

**Recursive function calls work exactly the same way**
New set of parameters and local variables for each (recursive) call

```
int factorial(int n){
  int result;
  if (n <= 1)
    result = 1;
  else
    result = n * factorial(n - 1);
  return result;
}
```

```
int main(void) {
  ...
  k = factorial(4);
  ...
}
```

```
 Trace
```
main k 24
```
```
Recursive & Base Cases
A recursive definition has two parts
One or more recursive cases where the function calls itself
One or more base cases that return a result without a recursive call

There must be at least one base case
Every recursive case must make progress towards a base case
Forgetting one of these rules is a frequent cause of errors with recursion

```
Recursive & Base Cases
Base case
Recursive caseint factorial(int n){
  int result;
  if (n <= 1)
    result = 1;
  else
    result = n * factorial(n - 1);
  return result;
}
```

```
Does This Run Forever?
Check:
Includes a base case? Yes
Recursive calls make progress? Hmmm...
Answer: Not known!!!
In tests, it always gets to the base case eventually, but nobody has been able to prove that this must be so!
```
```
```c
int f (int x) {
    if (x == 1)
        return 1;
    else if (x % 2 == 0)
        return 1 + f(x/2);
    else
        return 1 + f(3*x + 1);
}
```
**3N + 1 function**

\[
\begin{align*}
&f(5) = 1 + f(16) = 2 + f(8) = 3 + f(4) \\
&= 4 + f(2) = 5 + f(1) = 6 \\
&f(7) = 1 + f(22) = 2 + f(11) = 3 + f(34) \\
&= 4 + f(17) = 5 + f(52) = 6 + f(26) \\
&= 7 + f(13) = 8 + f(40) = 9 + f(20) \\
&= 10 + f(10) = 11 + f(5) = 12 + f(16) \\
&= 13 + f(8) = 14 + f(4) = 15 + f(2) \\
&= 16 + f(1) = 17
\end{align*}
\]

**Summary**

- Introduced Recursion
- Functions that call themselves
- Base and recursive cases
- Examples
  - Factorial
  - 3N+1 function
- An important concept with many uses