Chapter 9 - Recursion

Recursion

Learning Objectives - Recursion

- **Define recursion**
- **Identify the base case**
- **Identify the recursive pattern**

What is Recursion?

What is Recursion?

Solving a coding problem with functions involves breaking down the problem into smaller problems. When these smaller problems are variations of the larger problem (also know as self-similar), then recursion can be used. For example, the mathematical function factorial is self-similar. Five factorial (`5!`) is calculated as `5 * 4 * 3 * 2 * 1`. Mouse over the image below to see that `5!` is really just `5 * 4!`, and `4!` is really just `4 * 3!` and so on.

Because `5!` is self-similar, recursion can be used to calculate the answer. Recursive functions are functions that call themselves. Use the Code Visualizer to see how Python handles this recursive function.

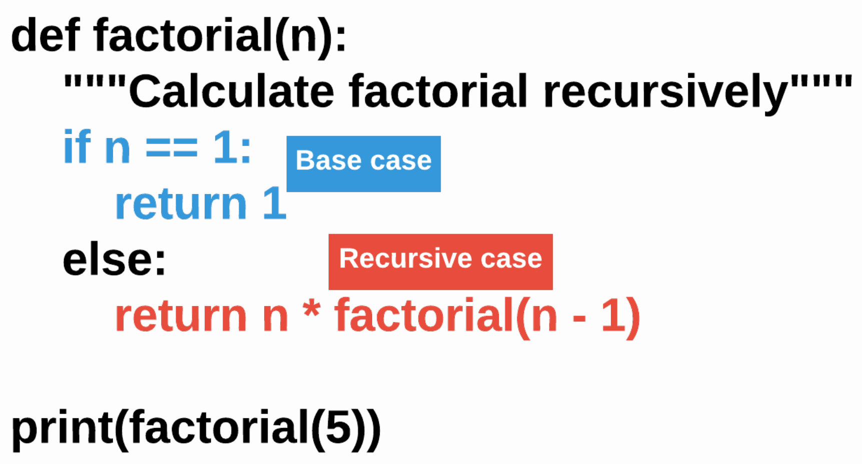
```python
def factorial(n):
    """Calculate factorial recursively"""
    if n == 1:
        return 1
    else:
        return n * factorial(n - 1)

print(factorial(5))
```

Code Visualizer

Recursion is an abstract and difficult topic, so it might be a bit hard to follow what is going on here. When `n` is 5, Python starts a multiplication problem of `5 * factorial(4)`. The function runs again and the multiplication problem becomes `5 * 4 * factorial(3)`. This continues until `n` is 1. Python returns the value `1`, and Python solves the multiplication problem `5 * 4 * 3 * 2 * 1`. The video below should help explain how `5!` is calculated recursively.

The Base Case

Each recursive function has two parts: the recursive case (where the function calls itself with a different parameter) and the base case (where the function stops calling itself and returns a value).



```python
def factorial(n):
    """Calculate factorial recursively"""
    if n == 1:
        return 1
    else:
        return n * factorial(n - 1)

print(factorial(5))
```

Cases for Recursion

The base case is the most important part of a recursive function. Without it, the function will never stop calling itself. Like an infinite loop, Python will stop the program with an error.

```python
def factorial(n):
    """Recursion without a base case"""
    return n * factorial(n - 1)

print(factorial(5))
```

Code Visualizer

Always start with the base case when creating a recursive function. Each time the function is called recursively, the program should get one step closer to the base case.

challenge

What happens if you:

- Add a base case for the `factorial` function?
- Change the print statement to `print(factorial(0))`?

Modify the base case so that `factorial(0)` does not result in an error. Test your new base case with a negative number.

▼ **Solution**

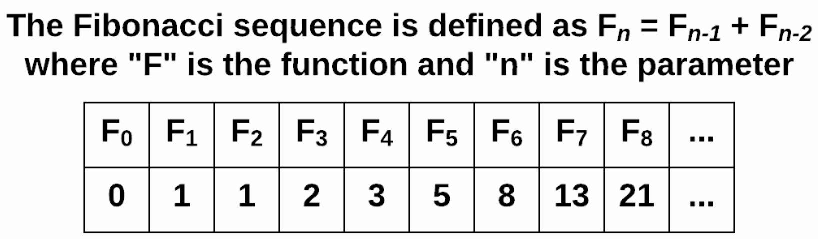
The factorial operation only works with positive integers. So the base case should be `if n &lt;= 0:`.

Code Visualizer

Fibonacci Sequence

Fibonacci Number

A Fibonacci number is a number in which the current number is the sum of the previous two Fibonacci numbers.



The Fibonacci sequence is defined as $F_n = F_{n-1} + F_{n-2}$
where "F" is the function and "n" is the parameter

F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	...
0	1	1	2	3	5	8	13	21	...

Fibonacci Sequence

Calculating a Fibonacci number is self-similar, which means it can be define with recursion. Setting the base case is important to avoid infinite recursion. When the number `n` is 0 the Fibonacci number is 0, and when `n` is 1 the Fibonacci number is 1. So if `n` is less than or equal to 1, then return `n`. That is the base case.

```python
def fibonacci(n):
    """Calculate Fibonacci numbers"""
    if n <= 1:
        return n
    else:
        return(fibonacci(n-1) + fibonacci(n-2))

print(fibonacci(3))
```

Code Visualizer

challenge

What happens if you:

- Change the print statment to `print(fibonacci(0))`?
- Change the print statment to `print(fibonacci(8))`?
- Change the print statment to `print(fibonacci(30))`?

▼ **Where is the code visualizer?**

The code visualizer will only step through your code 1,000 times. These recursive functions exceed this limit and generate an error message. Because of this, the code visualizer was removed.

Fibonacci Sequence

Fibonacci numbers are most often talked about as a sequence. The code below adds the functionality of printing a Fibonacci sequence of predetermined length.

```python
def fibonacci(n):
    """Calculate Fibonacci numbers"""
    if n <= 1:
        return n
    else:
        return(fibonacci(n-1) + fibonacci(n-2))

fibonacci_length = 10
for num in range(fibonacci_length):
    print(fibonacci(num))
```

Code Visualizer

challenge

What happens if you:

- Change `fibonacci_length` to 30?
- Change `fibonacci_length` to 50?

▼ **Why is Python timing out?**

The code written above is terribly inefficient. Each time through the loop, Python is calculating the same Fibonacci numbers again and again. When `num` is 1, Python calculates the Fibonacci numbers for 0 and 1. When `num` is 2, Python is calculating the Fibonacci numbers for 0, 1, and 2. Once `num` becomes large enough, it becomes too much work for Python to have to recalculate these large numbers over and over again. There is a more efficient way to do this by using a data structure called a dictionary. The idea is to store previously calculated Fibonacci numbers in the dictionary. So instead of recalculating the same numbers again and again, you can get these numbers from the dictionary. If a Fibonacci number is not in the dictionary, then calculate it and add it to the dictionary. Data structures are a bit beyond the scope of these lessons, but here is the code of a more efficient way to calculate and print the Fibonacci sequence. Copy and paste the code below into the IDE if you want to run it.

```python
fibcache = {} #dictionary of Fibonacci numbers

def fibonacci(n):
    """Check to see if a Fibonacci number has been calculated (in th

    If not, add it to the dictionary and return it.
    If yes, return the number from the dictionary."""
    if n not in fibcache.keys():
        fibcache[n] = _fibonacci(n)
    return fibcache[n]

def _fibonacci(n):
    """Calculate Fibonacci number"""
    if n <= 1:
        return n
    else:
        fib = fibonacci(n-1) + fibonacci(n-2)
        return fib

fibonacci_length = 90
for num in range(fibonacci_length):
    print(fibonacci(num))
```