Recursive Functions

**Definition:** A function is called recursive if the body of that function calls itself, either directly or indirectly.

**Implication:** Executing the body of a recursive function may require applying that function recursively.

---

Digit Sums

- If a number n is divisible by 9, then sum_digits(n) is also divisible by 9.
  - Useful for typo detection!

The sum of the digits of 2019 is 12.

Sum of these digits + This digit

That is, we can break the problem of summing the digits of 2019 into a smaller instance of the same problem, plus some extra stuff. We call this recursion.

---

Sum Digits Without a While Statement

```python
def split(n):
    """Split positive n into all but its last digit and its last digit."
    return n // 10, n % 10

def sum_digits(n):
    """Return the sum of the digits of positive integer n."""
    if n < 10:
        return n
    else:
        all_but_last, last = split(n)
        return sum_digits(all_but_last) + last
```

---

The Problem Within the Problem

The sum of the digits of 6 is 6.

Likewise for any one-digit (non-negative) number (i.e., ≤ 10).

The sum of the digits of 2019 is 12.

Sum of these digits + This digit

That is, we can break the problem of summing the digits of 2019 into a smaller instance of the same problem, plus some extra stuff. We call this recursion.

---

The Anatomy of a Recursive Function

- The def statement header is similar to other functions.
- Conditional statements check for base cases.
- Base cases are evaluated without recursive calls.
- Recursive cases are evaluated with recursive calls.

```python
def sum_digits(n):
    """Return the sum of the digits of positive integer n."""
    if n < 10:
        return n
    else:
        all_but_last, last = split(n)
        return sum_digits(all_but_last) + last
```

(Demo)
Recursion in Environment Diagrams

- The same function `fact` is called multiple times,
- Different frames keep track of the different arguments in each call,
- What `n` evaluates to depends upon the current environment,
- Each call to `fact` solves a simpler problem than the last; smaller `n`.

```
4! = 4 · 3 · 2 · 1 = 24
```

Verifying Recursive Functions

1. Verify the base case
2. Treat `fact` as a functional abstraction!
3. Assume that `fact(n-1)` is correct
4. Verify that `fact(n)` is correct

The Luhn Algorithm

- First: From the rightmost digit, which is the check digit, moving left, double the value of every second digit; if product of this doubling operation is greater than 9 (e.g., 7 · 2 = 14), then sum the digits of the products (e.g., 18: 1 + 8 = 9; 14: 1 + 4 = 5)
- Second: Take the sum of all the digits

```
1 3 8 7 4 3
2 3 6 7 8 3
```

The Luhn sum of a valid credit card number is a multiple of 10.
def sum_digits(n):
    """Return the sum of the digits of positive integer n."""
    if n < 10:
        return n
    else:
        all_but_last, last = split(n)
        return sum_digits(all_but_last) + last

Converting Recursion to Iteration
Can be tricky: Iteration is a special case of recursion.
Idea: Figure out what state must be maintained by the iterative function.

Converting Iteration to Recursion
More formulaic: Iteration is a special case of recursion.
Idea: The state of an iteration can be passed as arguments.

def sum_digits_iter(n):
    digit_sum = 0
    while n > 0:
        n, last = split(n)
        digit_sum = digit_sum + last
    return digit_sum

def sum_digits_rec(n, digit_sum):
    if n == 0:
        return digit_sum
    else:
        n, last = split(n)
        return sum_digits_rec(n, digit_sum + last)