Sum of digits using invariant programming

- Each recursive call handles one digit
- So we divide the initial number *n* into its digits:
  - $n = (d_{k-1}d_{k-2}\cdots d_2 d_1 d_0)$ *(where d_i is a digit)*

- Let's call the sum of digits function *s(n)*
- Then we can split the work in two parts:
  - $$s(n) = \underbrace{s(d_{k-1}d_{k-2}\cdots d_i)}_{s_i} + \underbrace{(d_{i-1} + d_{i-2} + \cdots + d_0)}_{a}$$

- s_i is the work still to do and *a* is the work already done
- To keep the formula true, we set $i' = i+1$ and $a' = a+d_i$
- When $i=k$ then $s_k=s(0)=0$ and therefore *a* is the answer

Example execution

- Example with $n=314159$:

$$s(n) = s(d_{k-1}d_{k-2}\cdots d_i) + (d_{i-1} + d_{i-2} + \cdots + d_0)$$

  - s(314159) = s(314159) + 0
  - s(314159) = s(31415) + 9
  - s(314159) = s(3141) + 14
  - s(314159) = s(314) + 15
  - s(314159) = s(31) + 19
  - s(314159) = s(3) + 20
  - s(314159) = s(0) + 23 = 0 + 23 = 23

Final program

- S = $(d_{k-1}d_{k-2}\cdots d_i)$
  A = $(d_{i-1} + d_{i-2} + \cdots + d_0)$

```
fun {SumDigits2 S A}
    if S==0 then A
    else
        {SumDigits2 (S div 10) A+(S mod 10)}
    end
end
```