- Consider again the example of adding a list of numbers $(a_1, \ldots, a_n)$.

code	Number of Additions	sum
$sum = 0$;	0	0
$sum = sum + a_1$	1	a_1
$sum = sum + a_2$	2	$a_1 + a_2$
$sum = sum + a_3$	3	$a_1 + a_2 + a_3$
$sum = sum + a_4$	4	$a_1 + a_2 + a_3 + a_4$
$\ldots$		
$sum = sum + a_n$	n	$a_1 + \ldots + a_n$

- To generalise: After k additions

$$sum = \sum_{j=1}^{j=k} a_j$$

- Note here the range of values for j: $1 \leq j \leq k$

- IMPORTANT: Need to distinguish between the number of additions here and the value of the loop counter when programming.

- Often the loop counter starts at 0.

- Also: If processing numbers stored in an array, the array indices start at 0 also.

- List of numbers: $(a_1, \ldots, a_n)$. Here the subscripts do not correspond to the array indices.

- Recall: A loop invariant is an assertion (condition) which holds both before and after executing the body of a loop.

- The loop invariant should be a statement about the variables that change within the loop

- From above:

$$a_1 + \ldots + a_n = \sum_{j=1}^{j=n} a_j$$

- When the addition finishes

$$sum = \sum_{j=1}^{j=n} a_j$$

  (postcondition)

- The invariant: After k additions (where $1 \le k \le n$) what will be stored in sum?

- The invariant:

$$sum = a_1 + \ldots + a_k \wedge k \le n = \sum_{j=1}^{j=k} a_j \wedge (k \le n)$$

- In this example we assume that the list of numbers in not empty.

- Suppose you have a very basic calculator which cannot do multiplication but can do addition. How could you use it to evaluate $3 * 4$?

- $3 * 4 = 3 + 3 + 3 + 3$

- That is, evaluate multiplication as repeated addition.

- Use the variable $result$ to store the result of the addition. After adding 0 numbers, $result = 0$

- After adding 1 number $result =$?

- After adding 2 numbers $result =$?

- After adding 3 numbers $result =$?

- After adding k numbers $result =$?

- Consider constructing a loop where a variable $result$ (initially set to 0) is changed at each iteration by adding 3 to it. (And the loop should iterate 4 times to evaluate $3 * 4$)

```
int i=0;
int result=0;
while(i<4)
{
    result=result+3; i=i+1;
}
```

- The loop follows the following execution:

code	iteration number(i)	result
$result = 0;$	0	0
$result = result + 3$	1	3
$result = result + 3$	2	6
$result = result + 3$	3	9
$result = result + 3$	4	12

- What is the invariant in this instance?

- Show that if the loop condition and the invariant holds at the start of an iteration of the loop then the invariant holds after the loop has executed.

  $\{i < 4 \wedge result = 3 * i \wedge i \leq 4\}$ (precondition: loop cond.$\wedge$ invariant)
  $result = result + 3; i = i + 1$
  $\{result = 3 * i \wedge i \leq 4\}$(postcondition: invariant)

- This is what you try to show when you apply the while rule for proving the correctness of loops

- Start with the last assignment and substitute its right hand side into the postcondition
  $result = 3 * (i + 1) \land i + 1 \leq 4$

- Now take the previous assignment and substitute its right hand side into this condition
  $result + 3 = 3 * (i + 1) \land i + 1 \leq 4$

- Simplify this and you get:
  $result = 3 * i \land i < 4$
  The precondition is stronger than (in this case as strong as) this condition

- Now suppose that the simple calculator you have can do multiplication but cannot evaluate powers e.g. x^y

- Construct a loop which will evaluate x^y using multiplication

- Identify the invariant for the loop

- Try to show that if the invariant and the loop condition holds before the loop body is executed, then the loop invariant holds afterwards.