

Iterative algorithms

- Looping constructs (e.g. while or for loops) lead naturally to **iterative** algorithms
- Can conceptualize as capturing computation in a set of “state variables” which update on each iteration through the loop

Iterative multiplication by successive additions

- Imagine we want to perform multiplication by successive additions:
 - To multiply a by b, add a to itself b times
- State variables:
 - i – iteration number, starts at b
 - result – current value of computation, starts at 0
- Update rules
 - $i \leftarrow i - 1$; stop when 0
 - $\text{result} \leftarrow \text{result} + a$

```
def iterMul(a, b):  
    result = 0  
    while b > 0:  
        result += a  
        b -= 1  
    return result
```

The diagram illustrates the iterative multiplication function `iterMul(a, b)`. It features three red annotations with arrows pointing to specific parts of the code:

- An arrow points from the text `result = result + a` to the `+=` operator in the line `result += a`.
- An arrow points from the text `update` to the `+=` operator in the line `result += a`.
- An arrow points from the text `b = b - 1` to the `-=` operator in the line `b -= 1`.

Other visual elements include a red circle around the parameter `b` in the function signature, a red oval around the initialization `result = 0`, and a red underline under the `return result` statement.