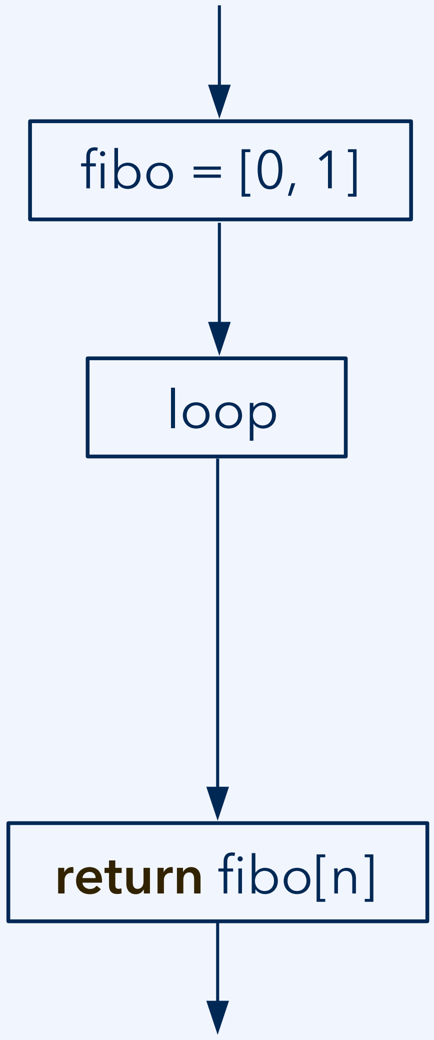
Tutorial 1.2 problem

CO2412

16th November 2021

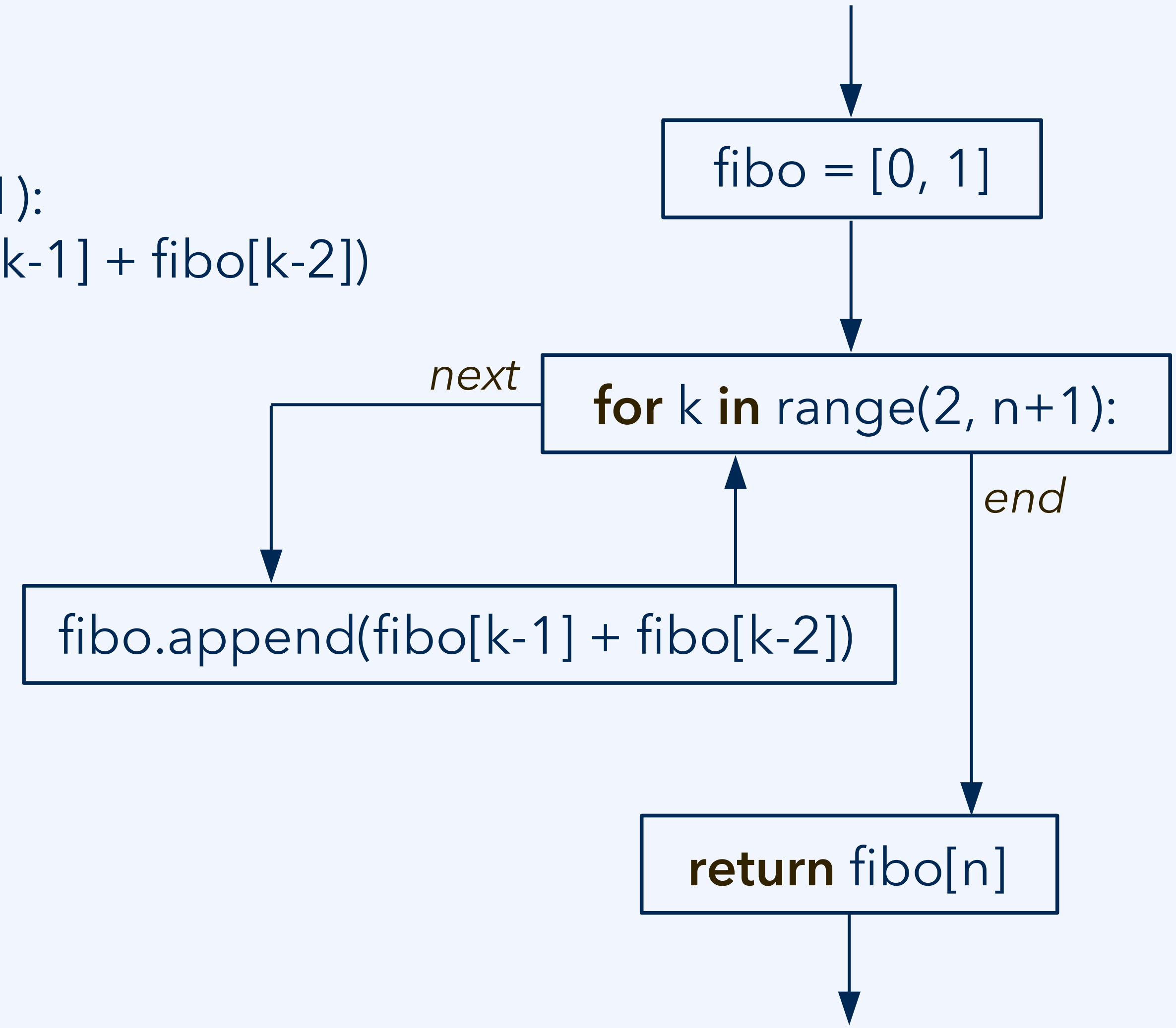
Iterative computation of Fibonacci numbers

```
def fibonacci_iter(n):
    fibo = [0, 1]
    for k in range(2, n+1):
        fibo.append(fibo[k-1] + fibo[k-2])
    return fibo[n]
```



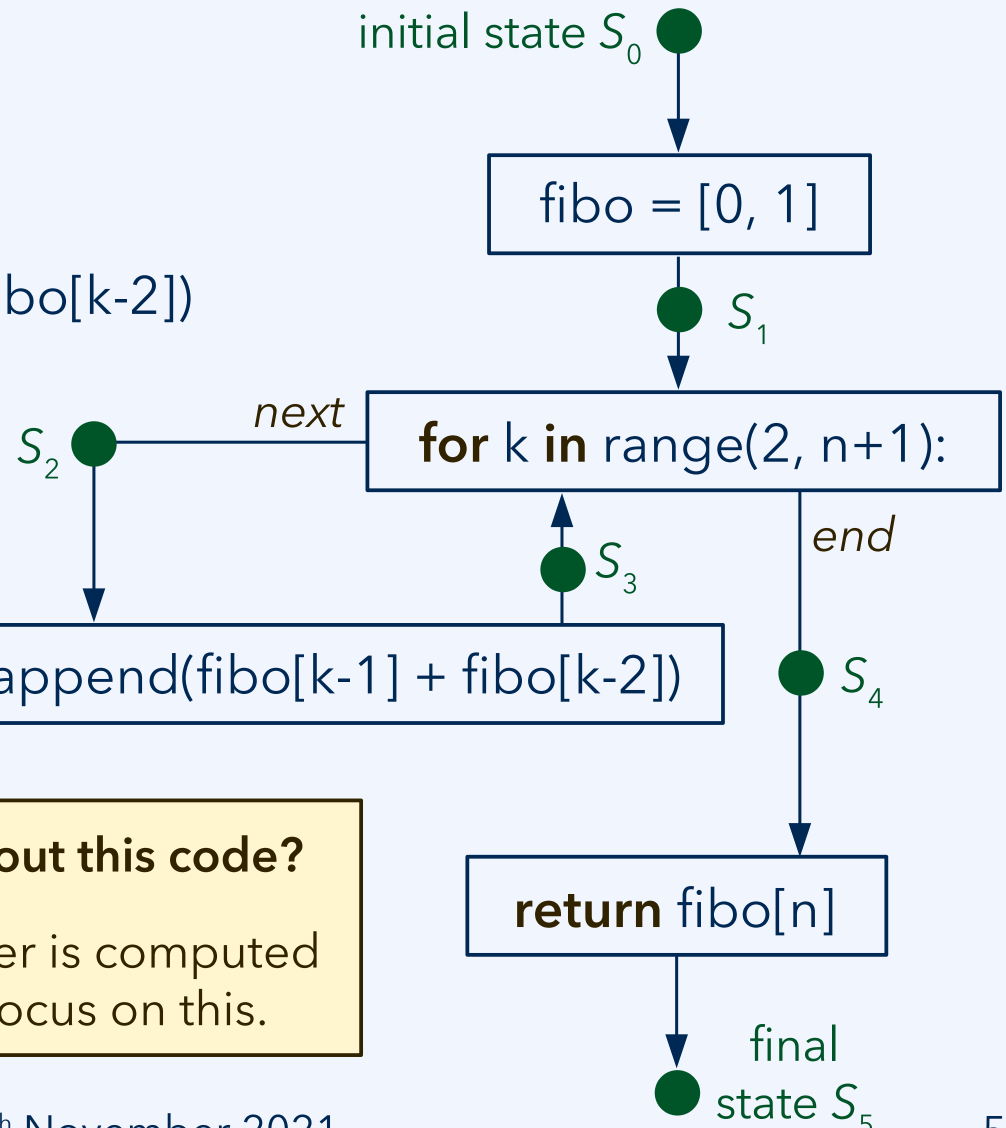
Iterative computation of Fibonacci numbers

```
def fibonacci_iter(n):
    fibo = [0, 1]
    for k in range(2, n+1):
        fibo.append(fibo[k-1] + fibo[k-2])
    return fibo[n]
```



Iterative computation of Fibonacci numbers

```
def fibonacci_iter(n):
    fibo = [0, 1]
    for k in range(2, n+1):
        fibo.append(fibo[k-1] + fibo[k-2])
    return fibo[n]
```



What do we want to prove about this code?

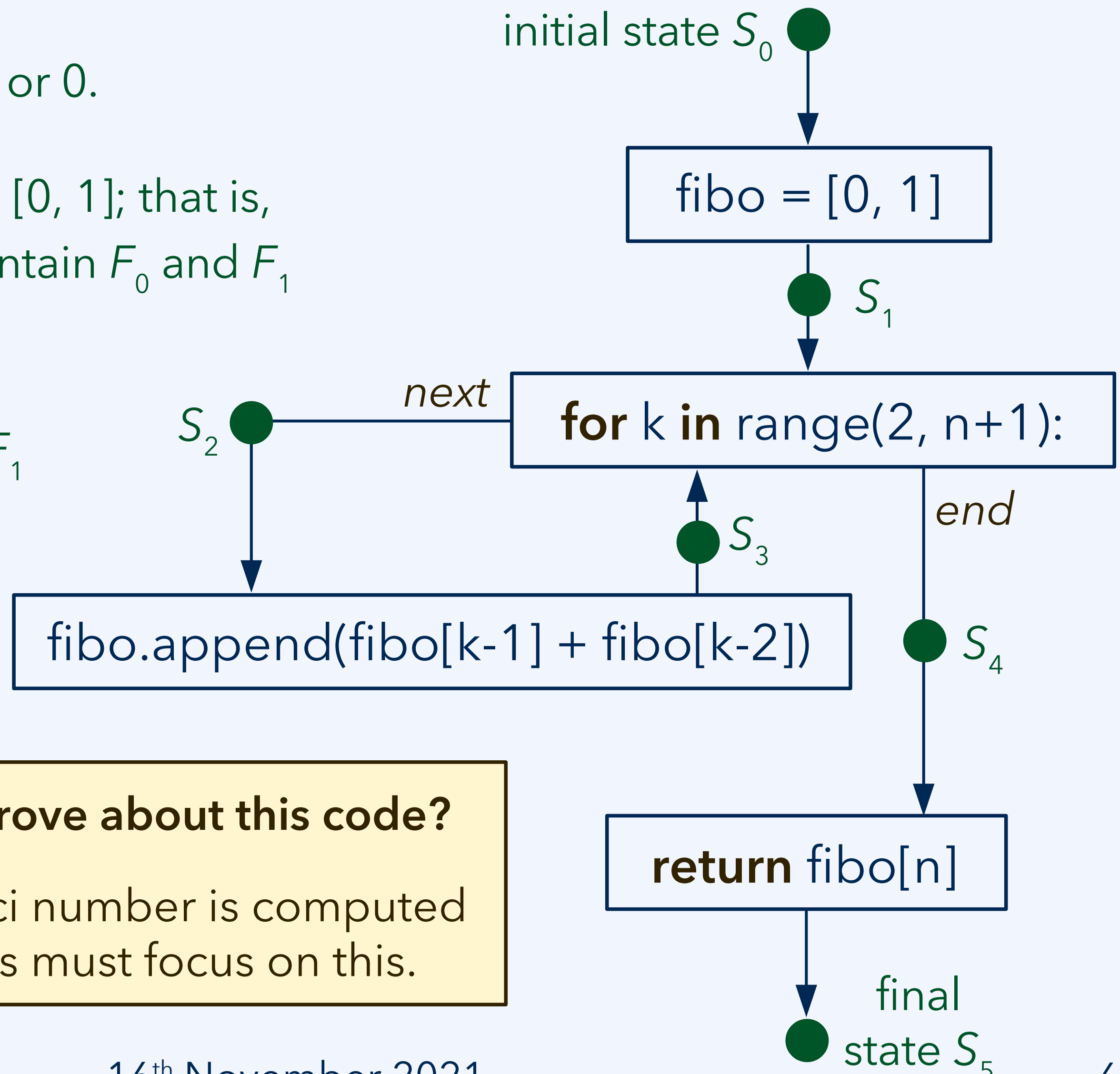
… that the *n*th Fibonacci number is computed correctly. Our analysis must focus on this.

Execution states: First iteration

S_0: n is a natural number or 0.

S_1: as above; also, fibo is [0, 1]; that is, fibo[0] and fibo[1] contain F_0 and F_1

S_2: k ≡ 2 and n ≥ 2; fibo contains F_0 and F_1



What do we want to prove about this code?

… that the *n*th Fibonacci number is computed correctly. Our analysis must focus on this.

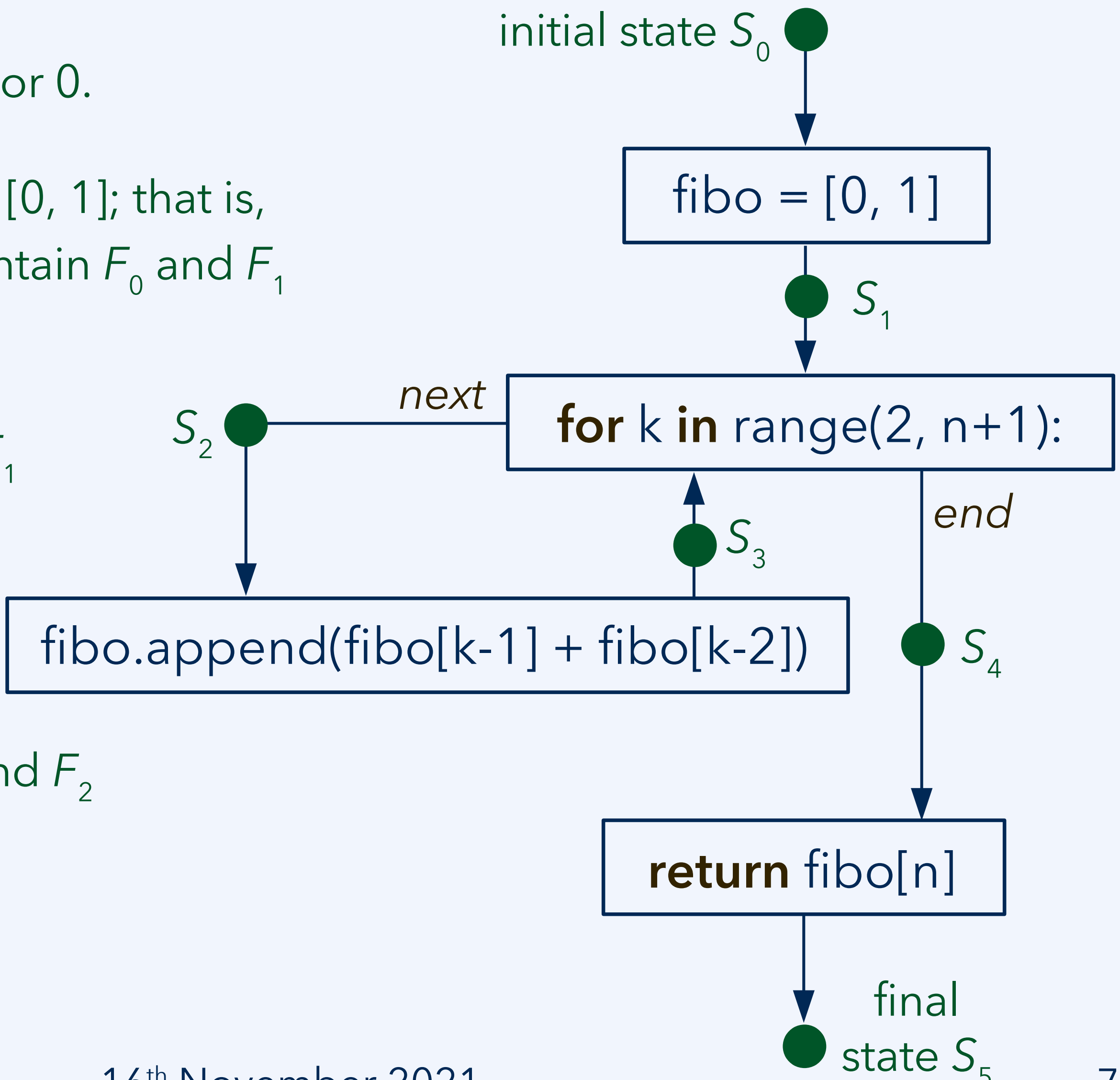
Execution states: First iteration

S_0: n is a natural number or 0.

S_1: as above; also, fibo is [0, 1]; that is, fibo[0] and fibo[1] contain F_0 and F_1

S_2: k ≡ 2 and n ≥ 2; fibo contains F_0 and F_1

S_3: k ≡ 2 and n ≥ 2; fibo contains F_0, F_1, and F_2



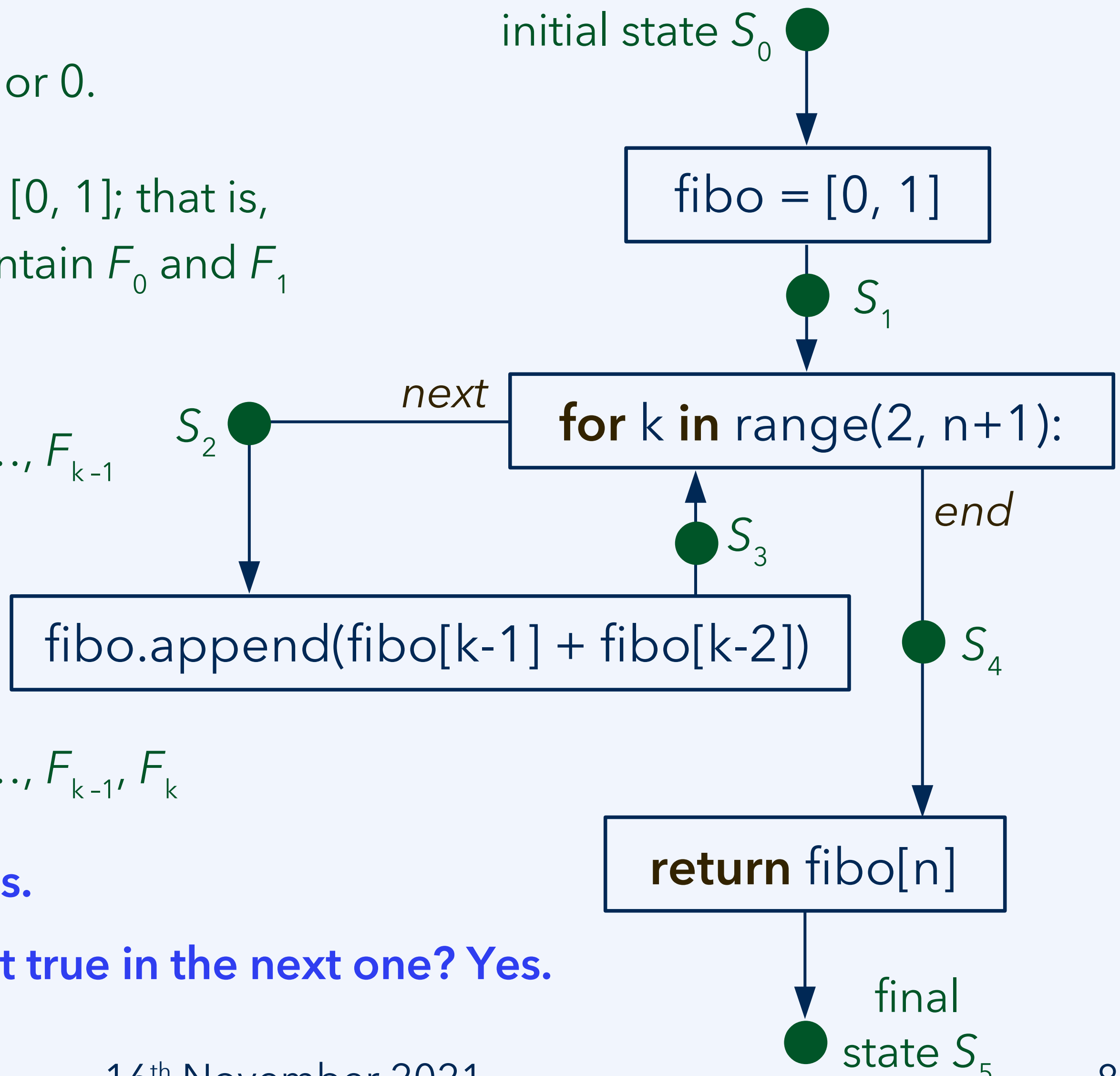
Execution states: Loop invariants

S_0: n is a natural number or 0.

S_1: as above; also, fibo is [0, 1]; that is,
fibo[0] and fibo[1] contain F_0 and F_1

S_2: $2 \leq k \leq n$;
fibo contains F_0 , F_1, …, F_{k-1}

S_3: $2 \leq k \leq n$;
fibo contains F_0 , F_1, …, F_{k-1}, F_k



Is it true the first time? Yes.

If true in one iteration, is it true in the next one? Yes.

Execution states and proof of correctness

S_0: n is a natural number or 0.

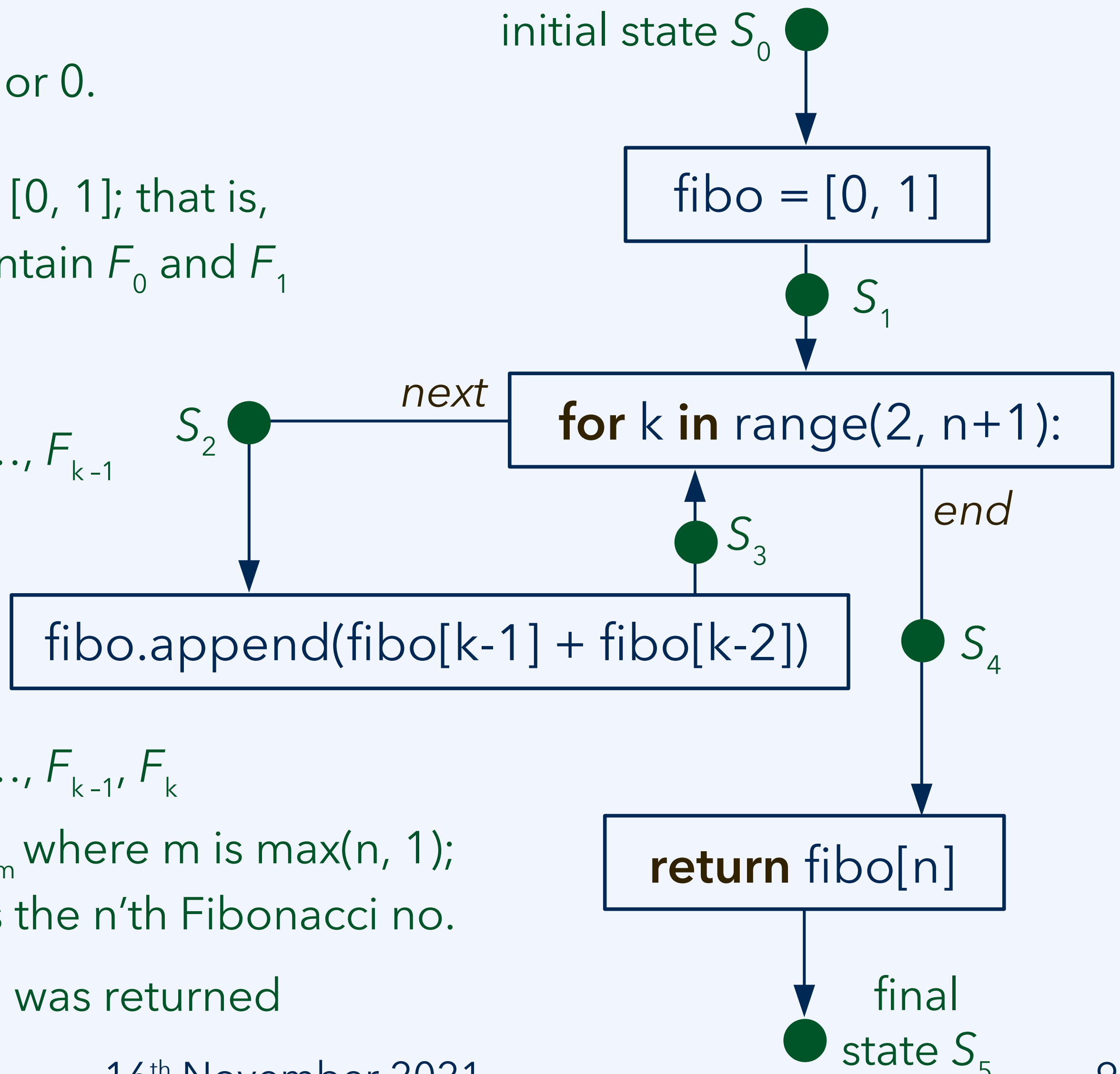
S_1: as above; also, fibo is [0, 1]; that is, fibo[0] and fibo[1] contain F_0 and F_1

S_2: $2 \le k \le n$;
fibo contains F_0 , F_1, …, F_{k-1}

S_3: $2 \le k \le n$;
fibo contains F_0 , F_1, …, F_{k-1}, F_k

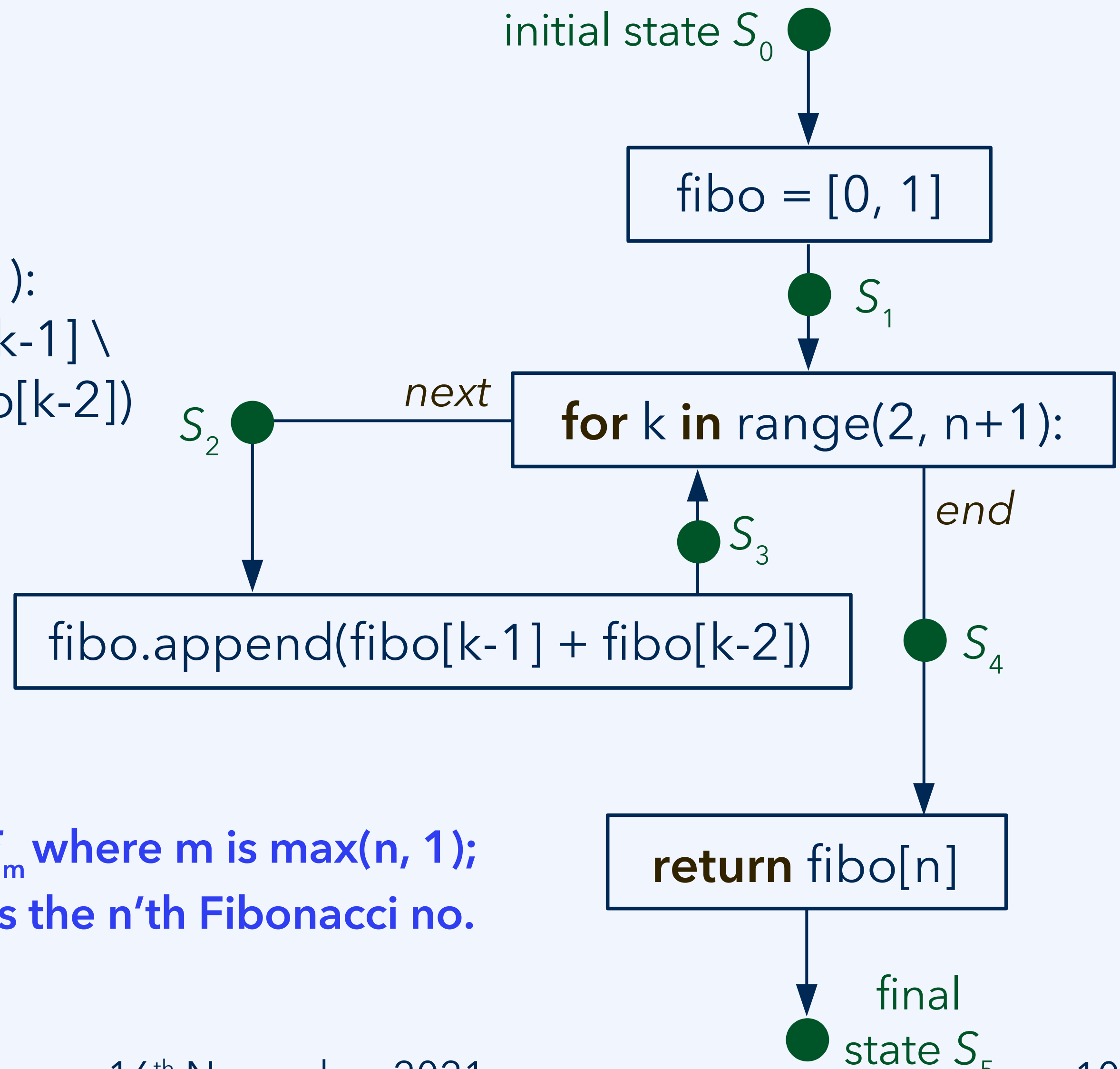
S_4: fibo contains F_0, …, F_m where m is max(n, 1);
in particular, fibo[n] is the n’th Fibonacci no.

S_5: the n’th Fibonacci no. was returned



Fibonacci code: Space efficiency

```
def fibonacci_iter(n):
    fibo = [0, 1]
    for k in range(2, n+1):
        fibo.append(fibo[k-1] \
                    + fibo[k-2])
    return fibo[n]
```



List size: O(*n*)

S_4: fibo contains F_0, …, F_m where m is max(n, 1); in particular, fibo[n] is the n'th Fibonacci no.

Fibonacci code: Memory optimization

```
def fibonacci_iter(n):
    fibo = [0, 1]
    for k in range(2, n+1):
        fibo.append(fibo[k-1] \
                    + fibo[k-2])
    return fibo[n]
```

List size: O(*n*)

S_4: fibo contains F_0, …, F_m where m is max(n, 1);
in particular, fibo[n] is the n'th Fibonacci no.

```
def fibonacci_iter(n):
    if n == 0:
        return 0
    F_k_minus_one, F_k = 0, 1  # k = 1
    for k in range(2, n+1):
        F_k_minus_two = F_k_minus_one
        F_k_minus_one = F_k
        F_k = F_k_minus_one \
              + F_k_minus_two
    return F_k  # k = n
```

Fibonacci code: Memory optimization

O(n) space code

```
def fibonacci_iter(n):
    fibo = [0, 1]
    for k in range(2, n+1):
        fibo.append(fibo[k-1] \
                    + fibo[k-2])
    return fibo[n]
```

List size: O(n)

S_4: fibo contains F_0, …, F_m where m is max(n, 1);
in particular, fibo[n] is the n'th Fibonacci no.

O(1) space code

```
def fibonacci_iter(n):
    if n == 0:
        return 0
    F_k_minus_one, F_k = 0, 1  # k = 1
    for k in range(2, n+1):
        F_k_minus_two = F_k_minus_one
        F_k_minus_one = F_k
        F_k = F_k_minus_one \
                + F_k_minus_two
    return F_k  # k = n
```

constant number of elementary variables

O(1) space