No, Maybe and Close Enough!

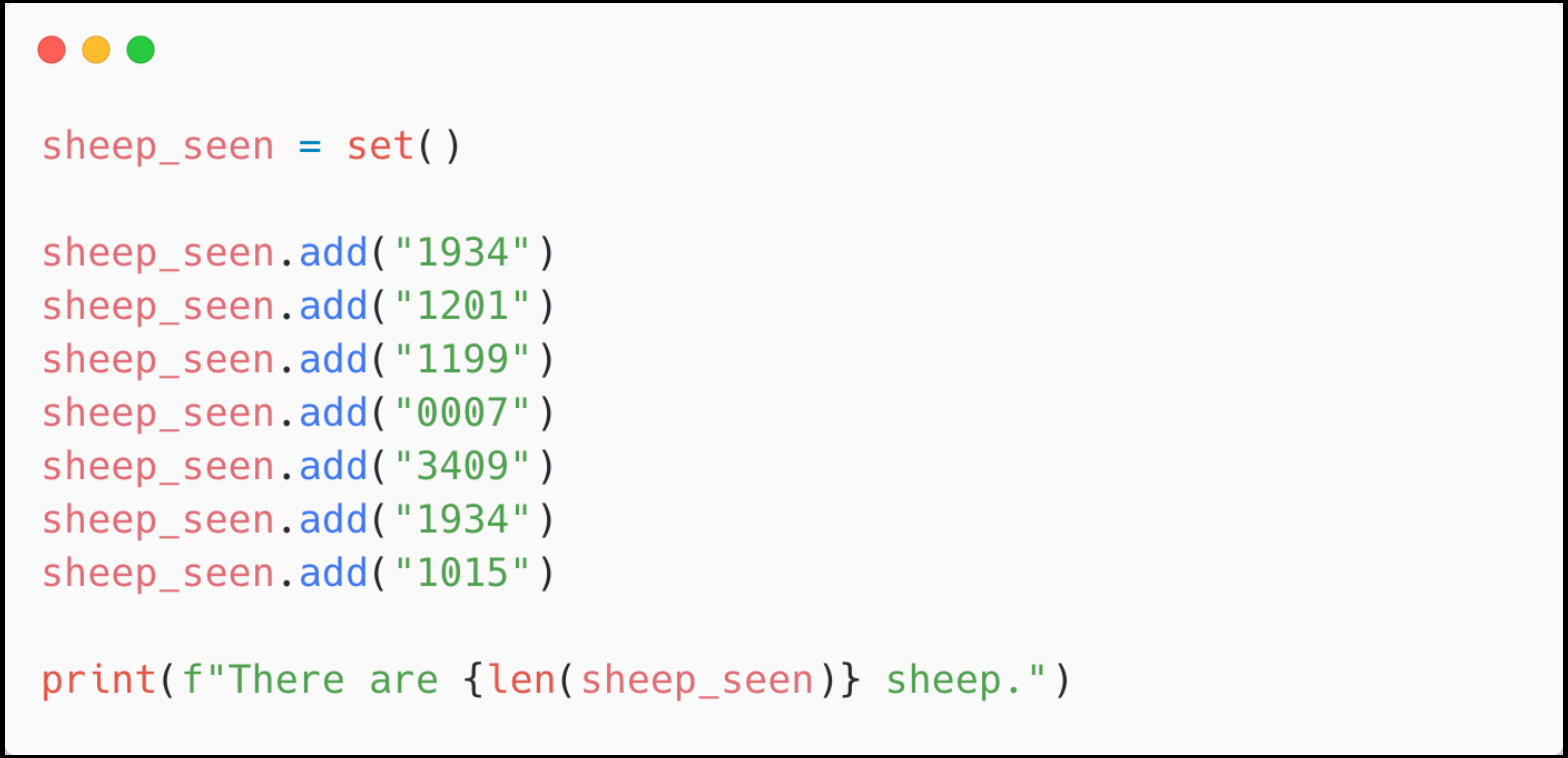
Probabilistic Data Structures with Python

Simon Prickett
https://simonprickett.dev

@simon_prickett

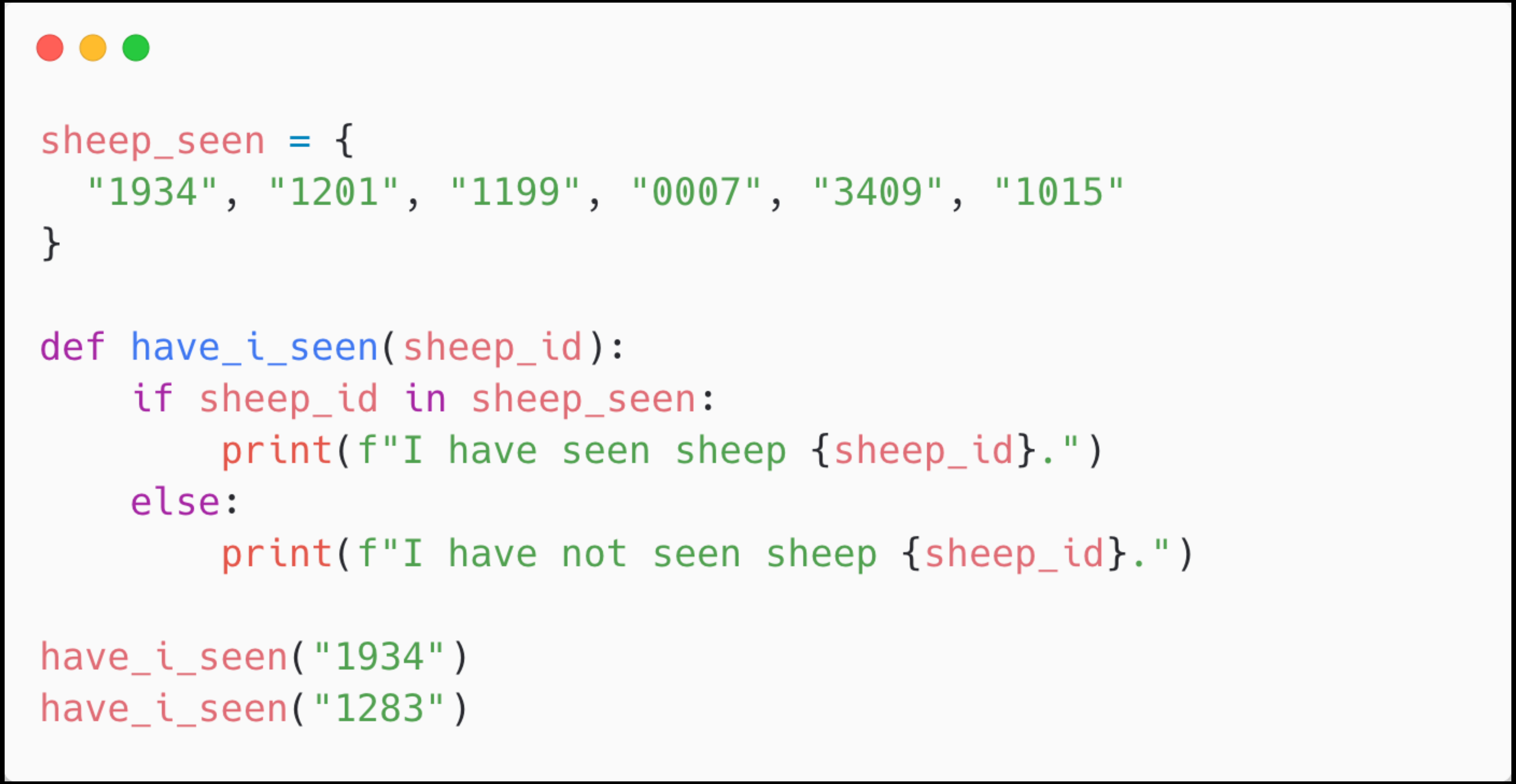
Counting Things…

How Many Sheep Have I Seen?



```python
sheep_seen = set()

sheep_seen.add("1934")
sheep_seen.add("1201")
sheep_seen.add("1199")
sheep_seen.add("0007")
sheep_seen.add("3409")
sheep_seen.add("1934")
sheep_seen.add("1015")

print(f"There are {len(sheep_seen)} sheep.")
```

Have I Seen This Particular Sheep?



```python
sheep_seen = {
  "1934", "1201", "1199", "0007", "3409", "1015"
}

def have_i_seen(sheep_id):
    if sheep_id in sheep_seen:
        print(f"I have seen sheep {sheep_id}.")
    else:
        print(f"I have not seen sheep {sheep_id}.")

have_i_seen("1934")
have_i_seen("1283")
```

Image by Christophe Becker
https://www.flickr.com/photos/christophebecker/39643834842

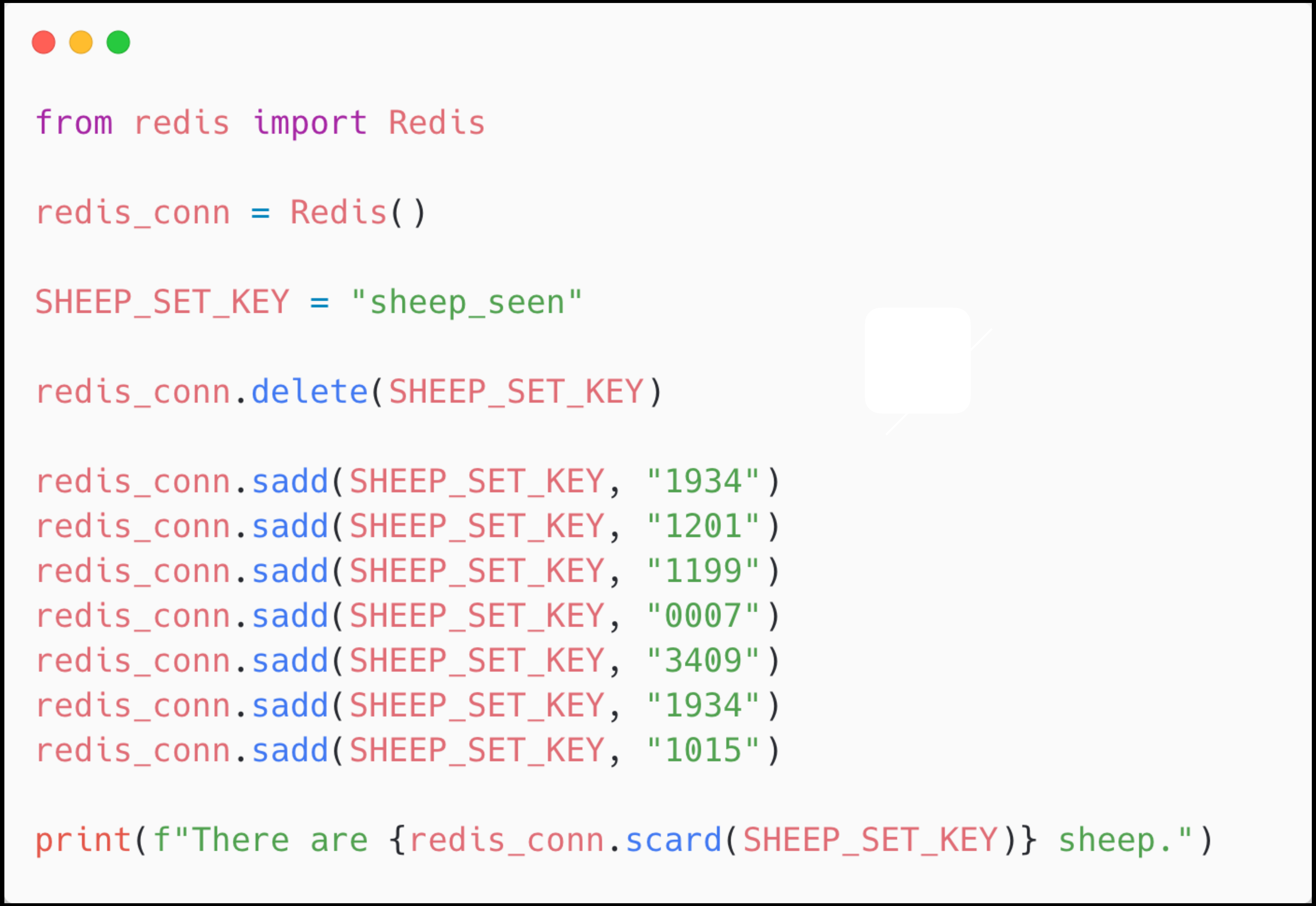
Hold on, is it really?

Go Big…

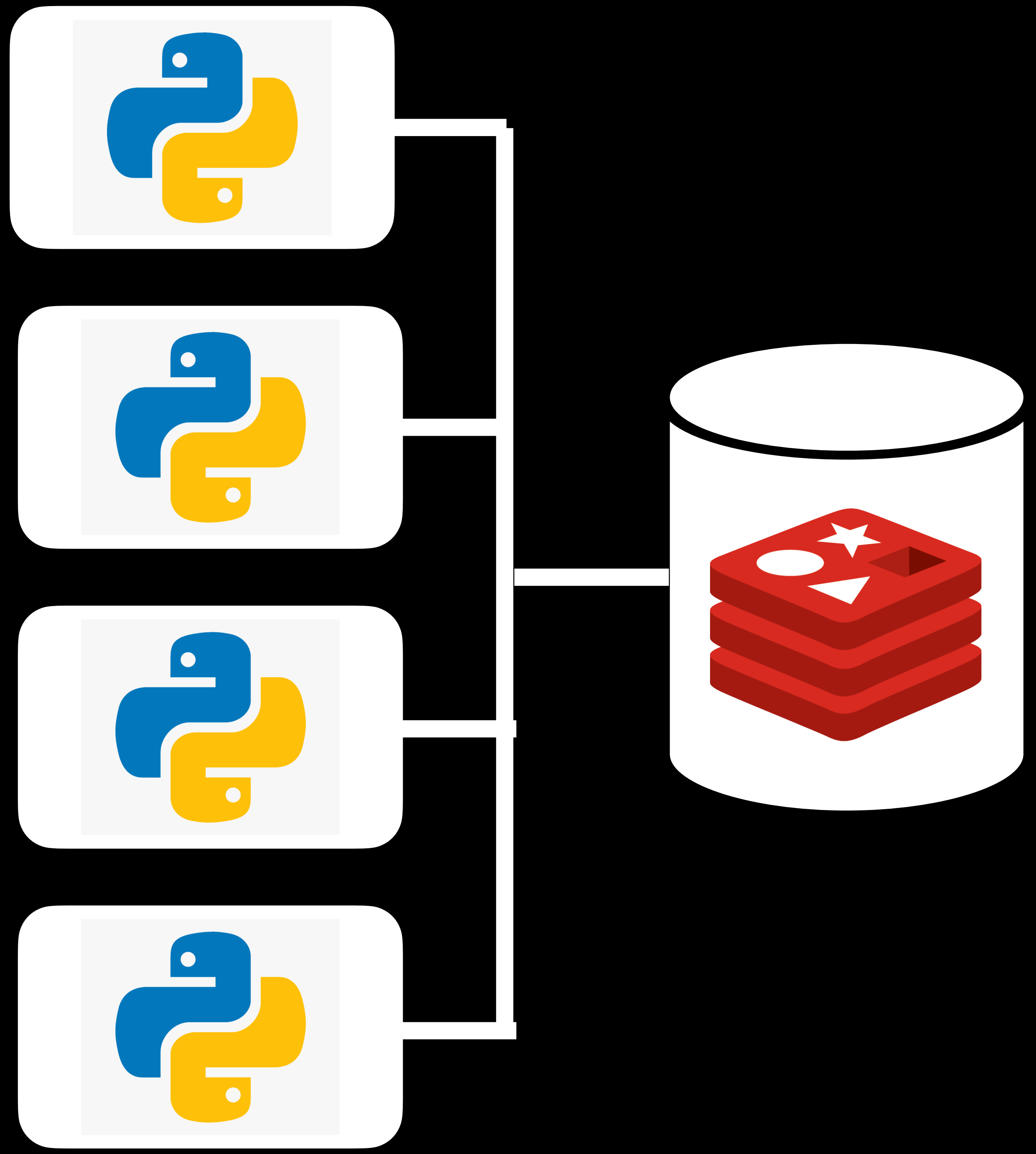
Problems at Scale

- Memory usage
- Horizontal scaling
- Exact counting gets expensive!

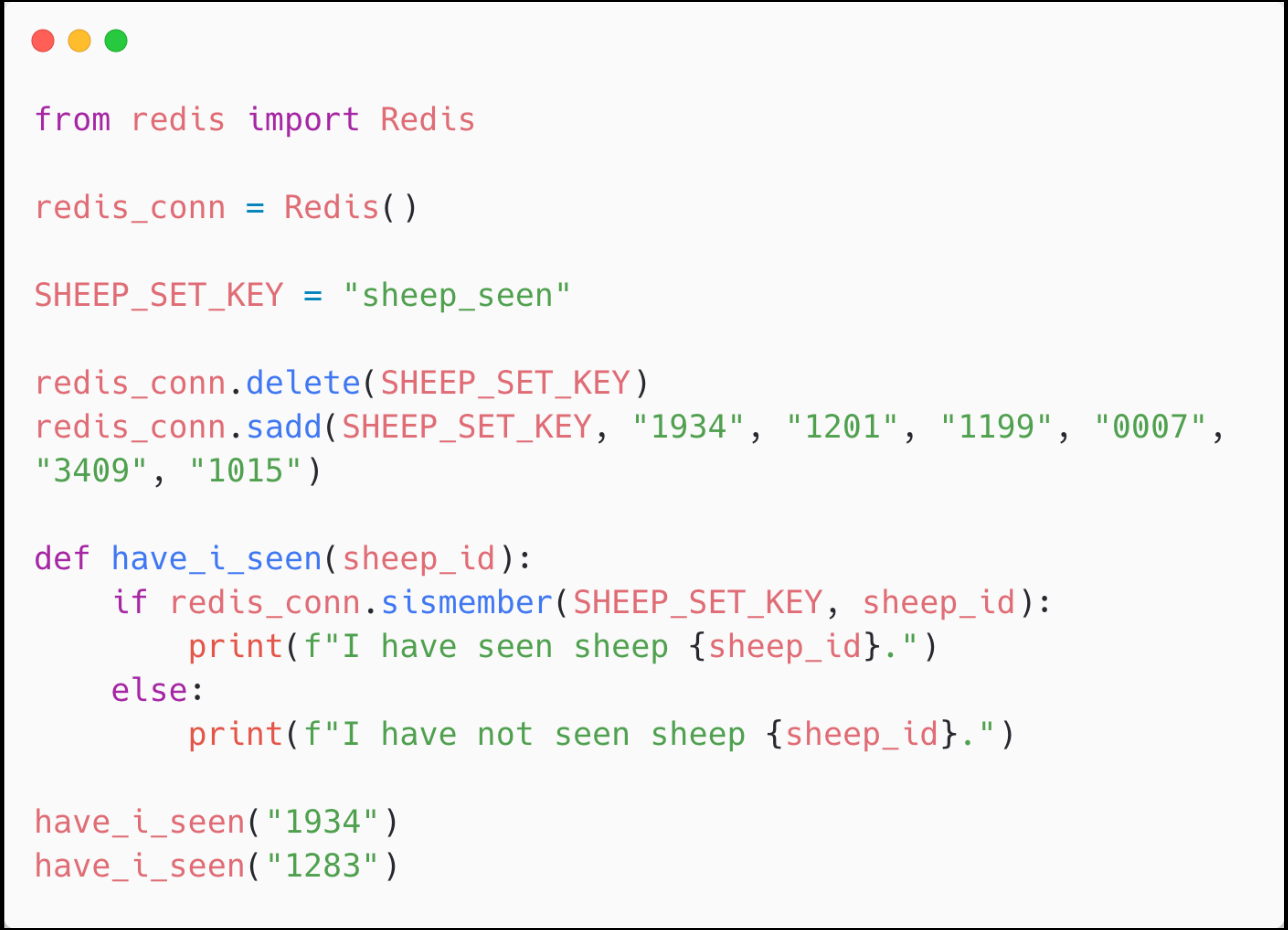
Use a Database: How Many Sheep?



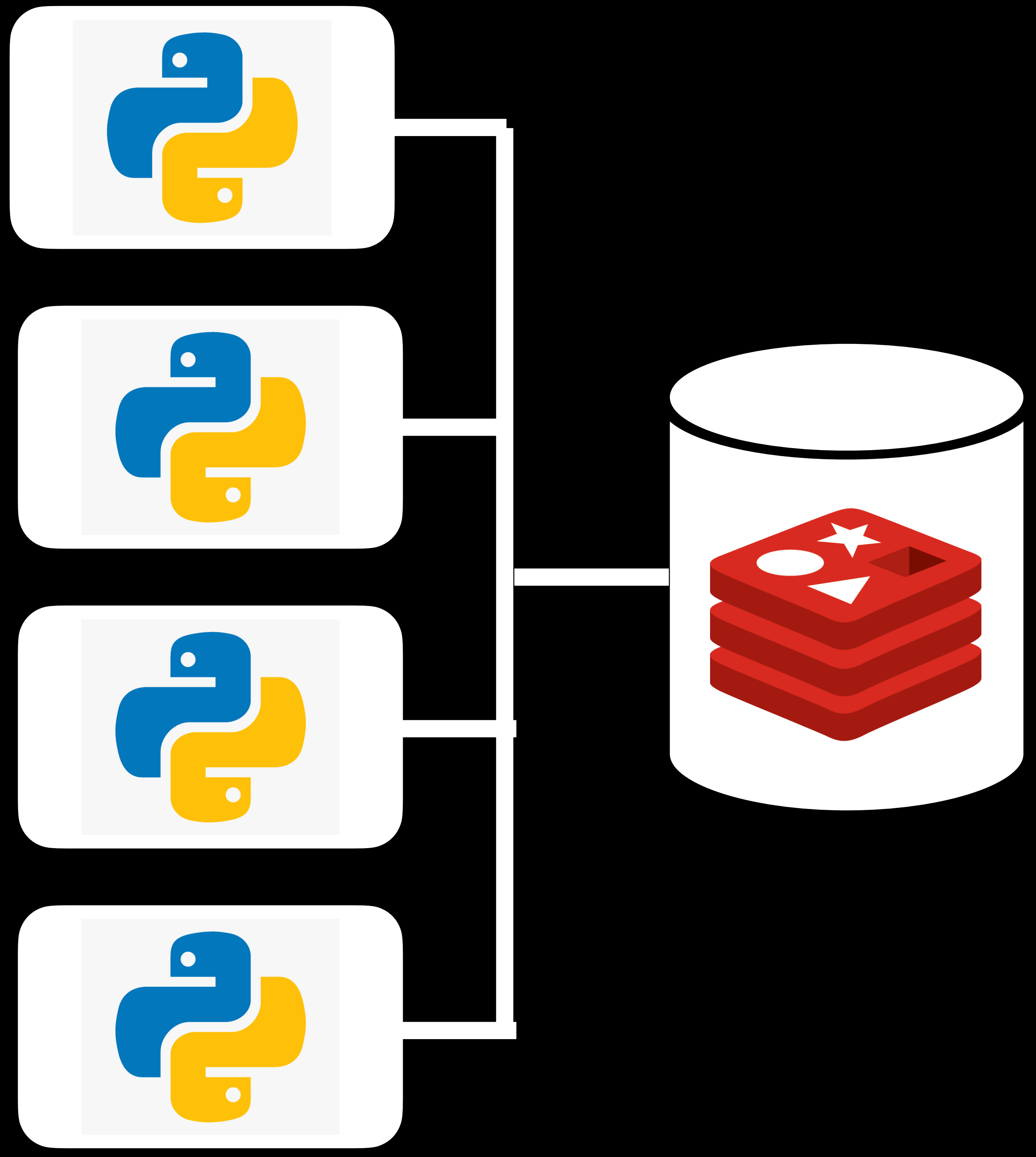
```
from redis import Redis

redis_conn = Redis()

SHEEP_SET_KEY = "sheep_seen"

redis_conn.delete(SHEEP_SET_KEY)

redis_conn.sadd(SHEEP_SET_KEY, "1934")
redis_conn.sadd(SHEEP_SET_KEY, "1201")
redis_conn.sadd(SHEEP_SET_KEY, "1199")
redis_conn.sadd(SHEEP_SET_KEY, "0007")
redis_conn.sadd(SHEEP_SET_KEY, "3409")
redis_conn.sadd(SHEEP_SET_KEY, "1934")
redis_conn.sadd(SHEEP_SET_KEY, "1015")

print(f"There are {redis_conn.scard(SHEEP_SET_KEY)} sheep.")
```



Use a Database: Have I Seen this Sheep?



```python
from redis import Redis

redis_conn = Redis()

SHEEP_SET_KEY = "sheep_seen"

redis_conn.delete(SHEEP_SET_KEY)
redis_conn.sadd(SHEEP_SET_KEY, "1934", "1201", "1199", "0007",
"3409", "1015")

def have_i_seen(sheep_id):
    if redis_conn.sismember(SHEEP_SET_KEY, sheep_id):
        print(f"I have seen sheep {sheep_id}.")
    else:
        print(f"I have not seen sheep {sheep_id}.")

have_i_seen("1934")
have_i_seen("1283")
```

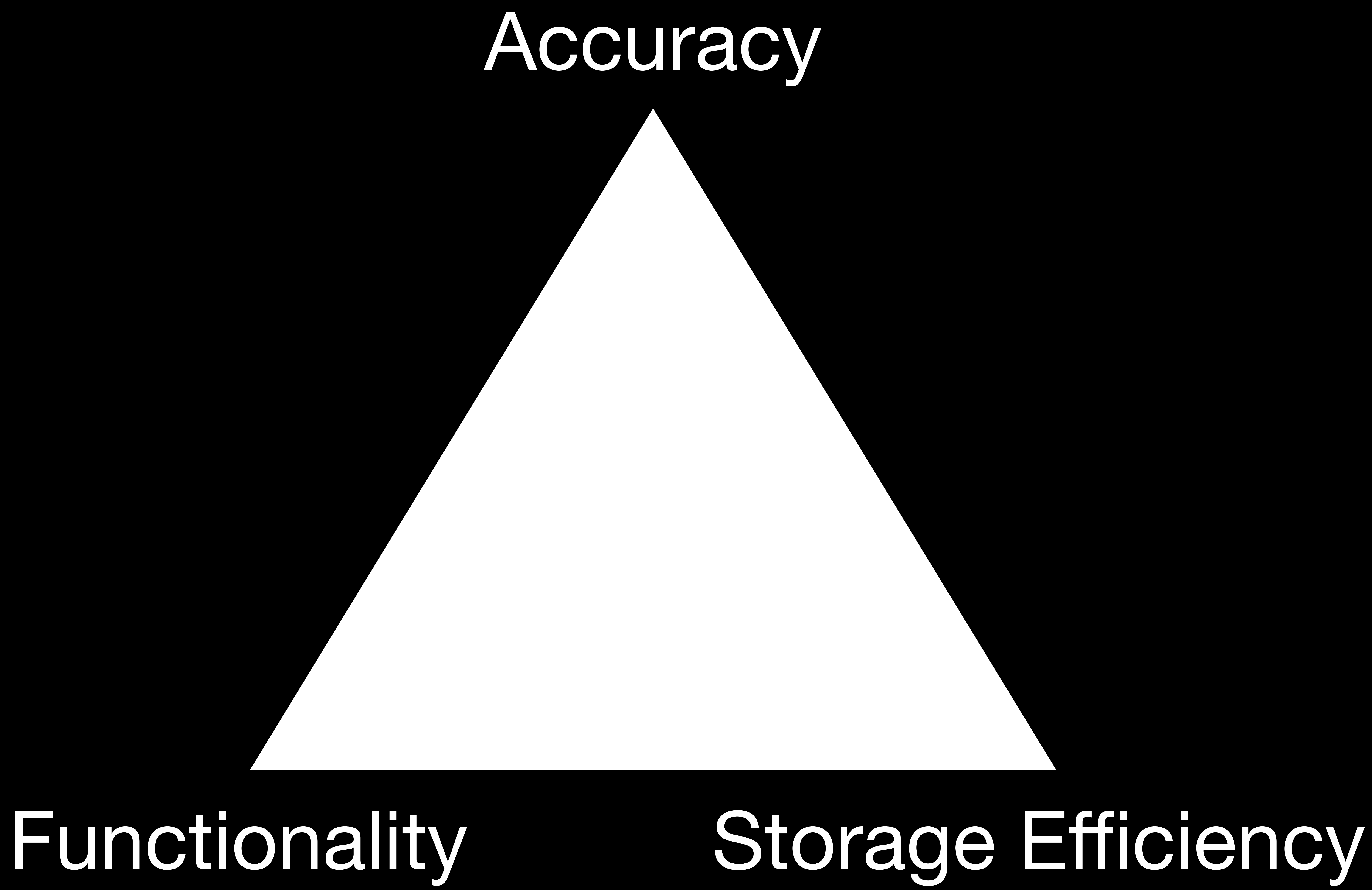


Tradeoff…

"a situational decision that involves diminishing or losing one quality, quantity, or property of a set or design in return for gains in other aspects." - Wikipedia

Image by cotaro70s
https://flickr.com/photos/cotaro70s/8670036813

Probabilistic Data Structures



Accuracy

Functionality

Storage Efficiency

Hyperloglog: Approximating Distinct Items

Benefits:

- Similar interface to a Set
- Much more space efficient than a Set
- Can't retrieve items, unlike a Set

Tradeoffs:

- Absolute Accuracy
- Can't retrieve items, unlike a Set
- Not built into Python, need an implementation
- Not built into many data stores

Hyperloglog: Algorithm

Add

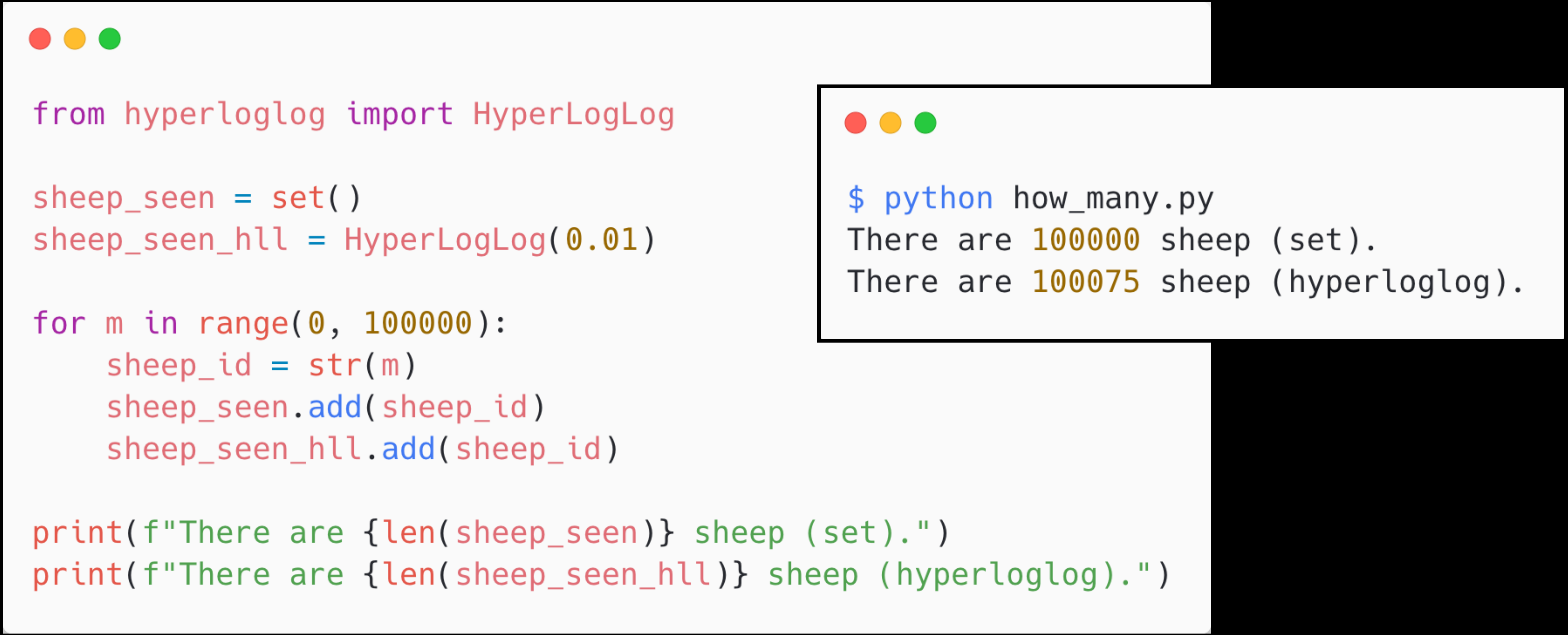
$$\begin{aligned} x &:= h(v) \\ j &:= 1 + \langle x_1, x_2, .., x_b \rangle_2 \\ w &:= x_{b+1} x_{b+2} \cdots \\ M[j] &:= \max(M[j], \rho(w)) \end{aligned}$$

Count

$$Z = \left(\sum_{j=1}^{m} 2^{-M[j]} \right)^{-1}$$

TL;DR Don't make your own, use a library or other implementation!

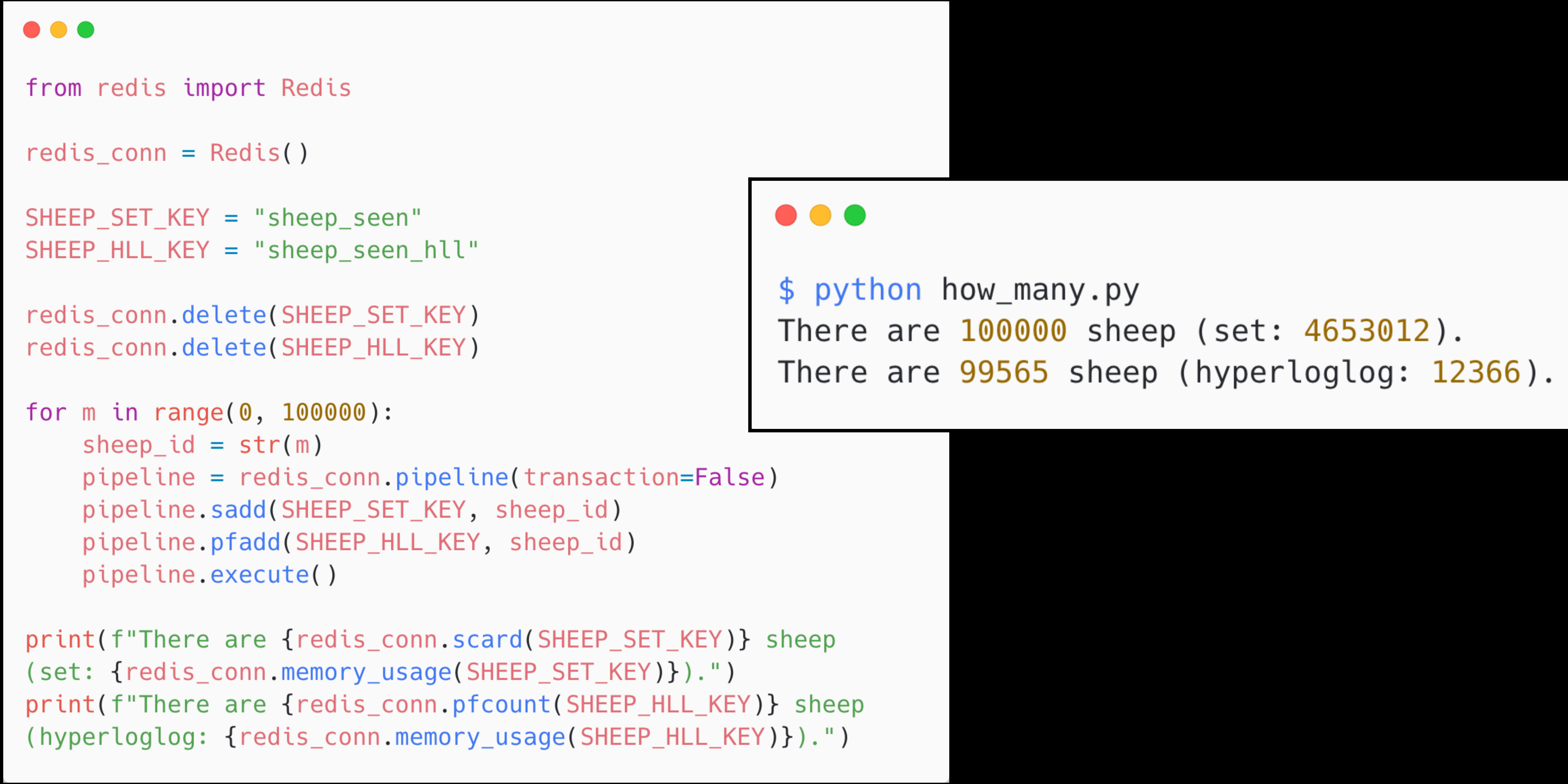
Approximately How Many Sheep Have I Seen?



```python
from hyperloglog import HyperLogLog

sheep_seen = set()
sheep_seen_hll = HyperLogLog(0.01)

for m in range(0, 100000):
    sheep_id = str(m)
    sheep_seen.add(sheep_id)
    sheep_seen_hll.add(sheep_id)

print(f"There are {len(sheep_seen)} sheep (set).")
print(f"There are {len(sheep_seen_hll)} sheep (hyperloglog).")
```

```
$ python how_many.py
There are 100000 sheep (set).
There are 100075 sheep (hyperloglog).
```

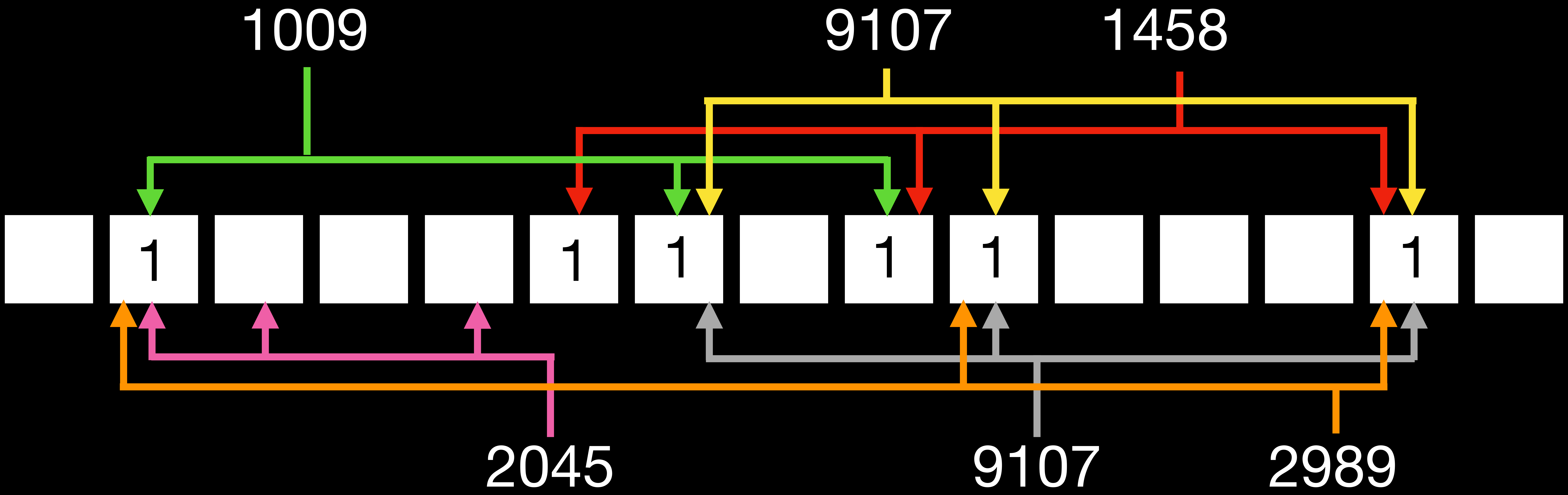
Redis: Approximately How Many Sheep Have I Seen?



```python
from redis import Redis

redis_conn = Redis()

SHEEP_SET_KEY = "sheep_seen"
SHEEP_HLL_KEY = "sheep_seen_hll"

redis_conn.delete(SHEEP_SET_KEY)
redis_conn.delete(SHEEP_HLL_KEY)

for m in range(0, 100000):
    sheep_id = str(m)
    pipeline = redis_conn.pipeline(transaction=False)
    pipeline.sadd(SHEEP_SET_KEY, sheep_id)
    pipeline.pfadd(SHEEP_HLL_KEY, sheep_id)
    pipeline.execute()

print(f"There are {redis_conn.scard(SHEEP_SET_KEY)} sheep
(set: {redis_conn.memory_usage(SHEEP_SET_KEY)}).")
print(f"There are {redis_conn.pfcount(SHEEP_HLL_KEY)} sheep
(hyperloglog: {redis_conn.memory_usage(SHEEP_HLL_KEY)}).")
```

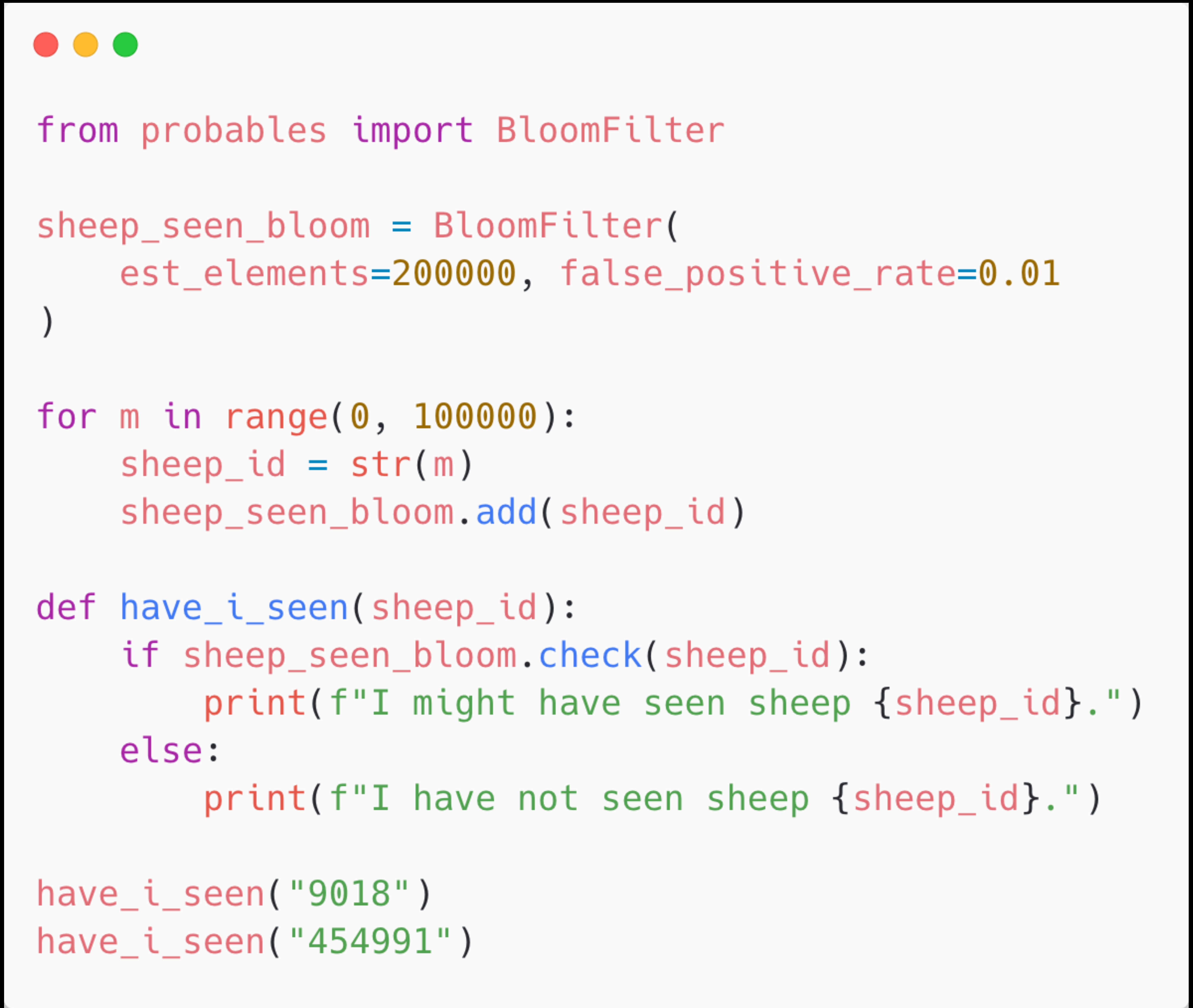
```
$ python how_many.py
There are 100000 sheep (set: 4653012).
There are 99565 sheep (hyperloglog: 12366).
```

Bloom Filter: Set Membership (No, Maybe)

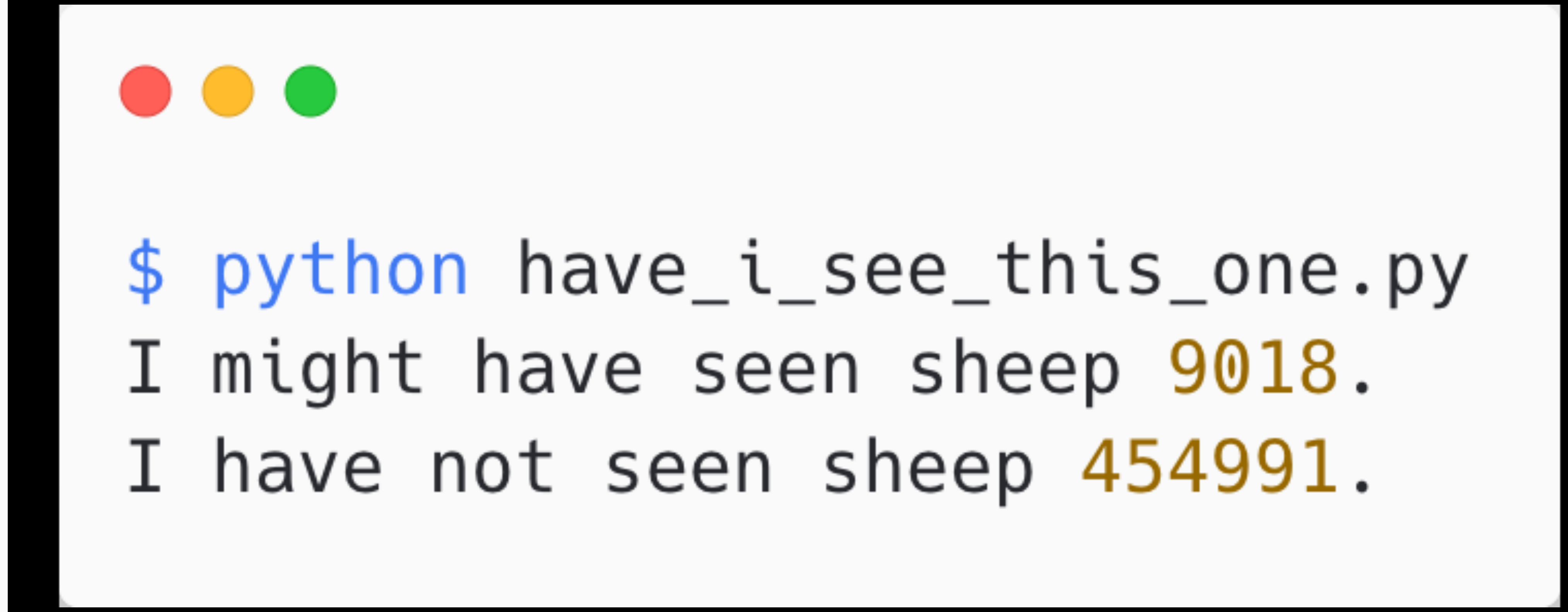


h1(sheepId) = 0…14
h2(sheepId) = 0…14
h3(sheepId) = 0…14

Have I Seen This Sheep (Maybe)?

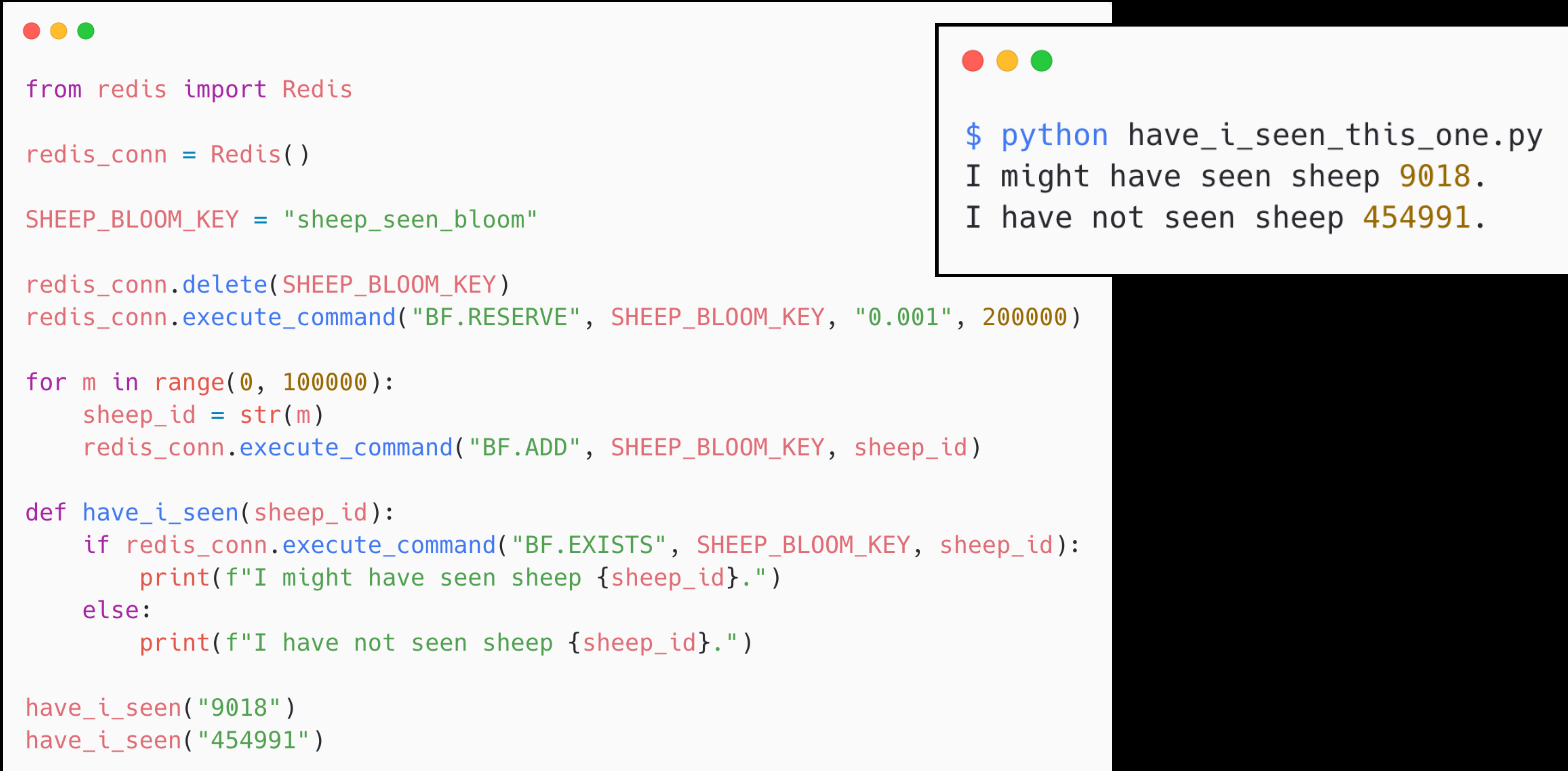


```python
from probables import BloomFilter

sheep_seen_bloom = BloomFilter(
    est_elements=200000, false_positive_rate=0.01
)

for m in range(0, 100000):
    sheep_id = str(m)
    sheep_seen_bloom.add(sheep_id)

def have_i_seen(sheep_id):
    if sheep_seen_bloom.check(sheep_id):
        print(f"I might have seen sheep {sheep_id}.")
    else:
        print(f"I have not seen sheep {sheep_id}.")

have_i_seen("9018")
have_i_seen("454991")
```



```
$ python have_i_see_this_one.py
I might have seen sheep 9018.
I have not seen sheep 454991.
```

Redis: Have I Seen This Sheep (Maybe)?



```python
from redis import Redis

redis_conn = Redis()

SHEEP_BLOOM_KEY = "sheep_seen_bloom"

redis_conn.delete(SHEEP_BLOOM_KEY)
redis_conn.execute_command("BF.RESERVE", SHEEP_BLOOM_KEY, "0.001", 200000)

for m in range(0, 100000):
    sheep_id = str(m)
    redis_conn.execute_command("BF.ADD", SHEEP_BLOOM_KEY, sheep_id)

def have_i_seen(sheep_id):
    if redis_conn.execute_command("BF.EXISTS", SHEEP_BLOOM_KEY, sheep_id):
        print(f"I might have seen sheep {sheep_id}.")
    else:
        print(f"I have not seen sheep {sheep_id}.")

have_i_seen("9018")
have_i_seen("454991")
```

```
$ python have_i_seen_this_one.py
I might have seen sheep 9018.
I have not seen sheep 454991.
```

When to use Probabilistic Data Structures

- If an approximate count is good enough (hyperloglog)
- If it's OK to have some false positives (Bloom Filter)
- When you don't need to retrieve the original data from the data structure
- When working with large data sets where exact strategies aren't practical

Thank You!

github.com/simonprickett/python-probabilistic-data-structures

Simon Prickett
@simon_prickett

https://simonprickett.dev