Sets and Dictionaries

Examples

How early was each kind of bird seen?

How early was each kind of bird seen?

```
2010-07-03    05:38    loon
2010-07-03    06:02    goose
2010-07-03    06:07    loon
2010-07-04    05:09    ostrich   # hallucinating?
2010-07-04    05:29    loon
     ⋮          ⋮       ⋮
```

How early was each kind of bird seen?

```
2010-07-03    05:38    loon
2010-07-03    06:02    goose
2010-07-03    06:07    loon
2010-07-04    05:09    ostrich   # hallucinating?
2010-07-04    05:29    loon
     ⋮          ⋮       ⋮
```

Want the minimum of all times associated with a bird

How early was each kind of bird seen?

```
2010-07-03    05:38    loon
2010-07-03    06:02    goose
2010-07-03    06:07    loon
2010-07-04    05:09    ostrich   # hallucinating?
2010-07-04    05:29    loon
     ⋮          ⋮       ⋮
```

Want the minimum of all times associated with a bird

Use bird name as dictionary key

How early was each kind of bird seen?

```
2010-07-03    05:38    loon
2010-07-03    06:02    goose
2010-07-03    06:07    loon
2010-07-04    05:09    ostrich   # hallucinating?
2010-07-04    05:29    loon
     ⋮          ⋮       ⋮
```

Want the minimum of all times associated with a bird

Use bird name as dictionary key

And earliest observation time as value

```
def read_observations(filename):
  '''Read data, return [(date, time, bird)...].'''

  reader = open(filename, 'r')
  result = []

  for line in reader:
    fields = line.split('#')[0].strip().split()
    assert len(fields) == 3, 'Bad line "%s"' % line
    result.append(fields)

return result
```

```
def read_observations(filename):
  '''Read data, return [(date, time, bird)...].'''

  reader = open(filename, 'r')
  result = []

  for line in reader:
    fields = line.split('#')[0].strip().split()
    assert len(fields) == 3, 'Bad line "%s"' % line
    result.append(fields)

return result
```

Setup

```
def read_observations(filename):
  '''Read data, return [(date, time, bird)...].'''

  reader = open(filename, 'r')
  result = []

  for line in reader:
    fields = line.split('#')[0].strip().split()
    assert len(fields) == 3, 'Bad line "%s"' % line
    result.append(fields)

return result
```

Get data from each line of the file

```
def read_observations(filename):
  '''Read data, return [(date, time, bird)...].'''

  reader = open(filename, 'r')
  result = []

  for line in reader:
    fields = line.split('#')[0].strip().split()
    assert len(fields) == 3, 'Bad line "%s"' % line
    result.append(fields)

return result
```

Check that the data might be right

```
def read_observations(filename):
  '''Read data, return [(date, time, bird)...].'''

  reader = open(filename, 'r')
  result = []

  for line in reader:
    fields = line.split('#')[0].strip().split()
    assert len(fields) == 3, 'Bad line "%s"' % line
    result.append(fields)

return result
```

Store it

```
def earliest_observation(data):
  '''How early did we see each bird?'''

  result = {}
  for (date, time, bird) in data:
    if bird not in result:
      result[bird] = time
    else:
      result[bird] = min(result[bird], time)

return result
```

```
def earliest_observation(data):
  '''How early did we see each bird?'''

  result = {}
  for (date, time, bird) in data:
    if bird not in result:
      result[bird] = time
    else:
      result[bird] = min(result[bird], time)

return result
```

Setup

```
def earliest_observation(data):
  '''How early did we see each bird?'''

  result = {}
  for (date, time, bird) in data:
    if bird not in result:
      result[bird] = time
    else:
      result[bird] = min(result[bird], time)

return result
```

Process each tuple in turn

```
def earliest_observation(data):
  '''How early did we see each bird?'''

  result = {}
  for (date, time, bird) in data:
    if bird not in result:
      result[bird] = time
    else:
      result[bird] = min(result[bird], time)

return result
```

First sighting, so this must be earliest time

```
def earliest_observation(data):
  '''How early did we see each bird?'''

  result = {}
  for (date, time, bird) in data:
    if bird not in result:
      result[bird] = time
    else:
      result[bird] = min(result[bird], time)

return result
```

Subsequent sighting, so take minimum

What birds were seen on each day?

What birds were seen on each day?

Very similar structure...

What birds were seen on each day?

Very similar structure...

...but use a set to record one or more birds,

rather than taking the minimum time

```
def birds_by_date(data):
  '''Which birds were seen on each day?'''

  result = {}
  for (date, time, bird) in data:
    if date not in result:
      result[date] = set()
    result[date].add(bird)

return result
```

```
def birds_by_date(data):
  '''Which birds were seen on each day?'''

  result = {}
  for (date, time, bird) in data:
    if date not in result:
      result[date] = set()
    result[date].add(bird)

return result
```

Setup

```
def birds_by_date(data):
  '''Which birds were seen on each day?'''

  result = {}
  for (date, time, bird) in data:
    if date not in result:
      result[date] = set()
    result[date].add(bird)

return result
```

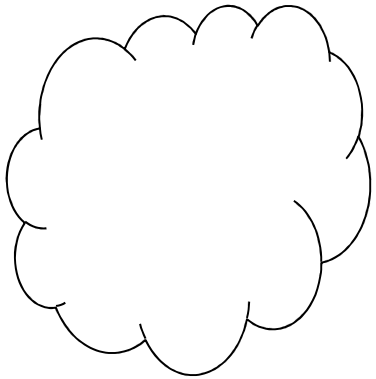
Process each tuple in turn

```
def birds_by_date(data):
  '''Which birds were seen on each day?'''

  result = {}
  for (date, time, bird) in data:
    if date not in result:
      result[date] = set()
    result[date].add(bird)

return result
```

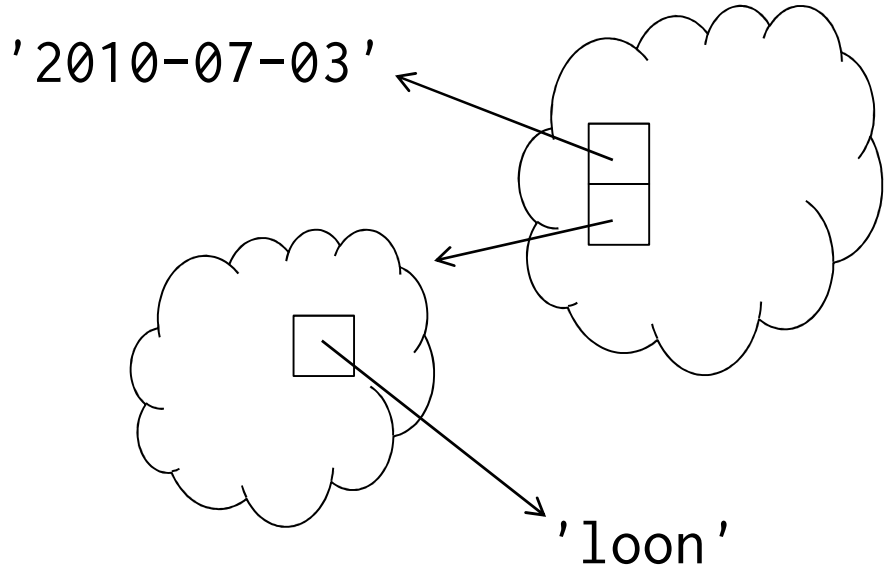
Make sure there is space to record this bird

```
def birds_by_date(data):
  '''Which birds were seen on each day?'''

  result = {}
  for (date, time, bird) in data:
    if date not in result:
      result[date] = set()
    result[date].add(bird)

return result
```

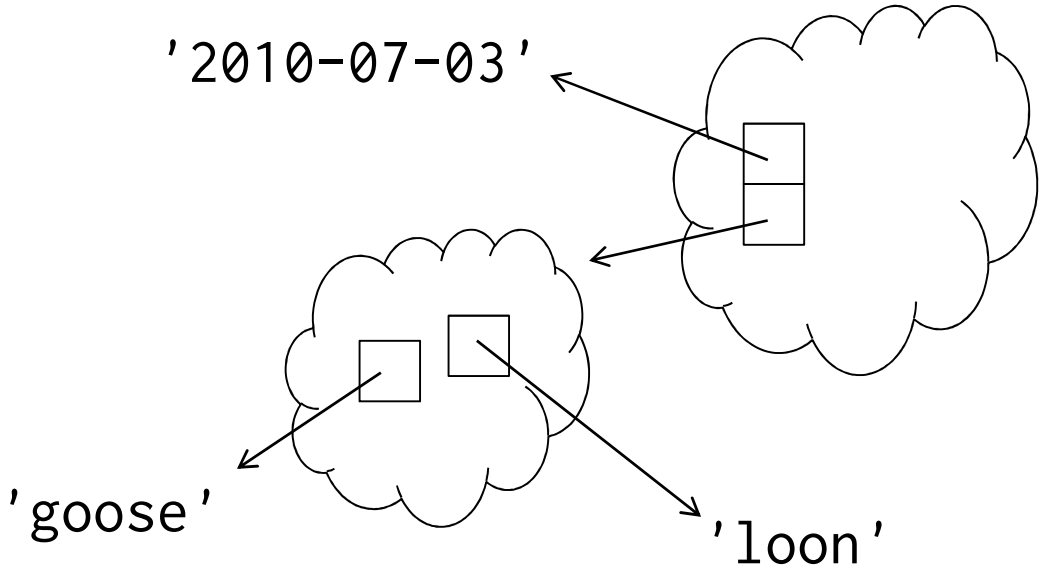
Record this bird



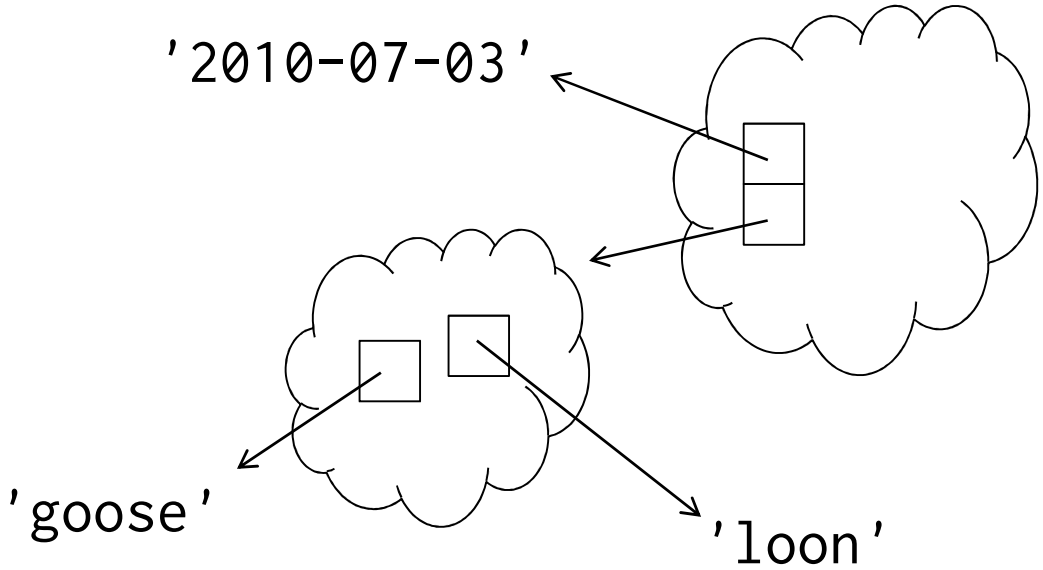
```
2010-07-03    05:38    loon
```



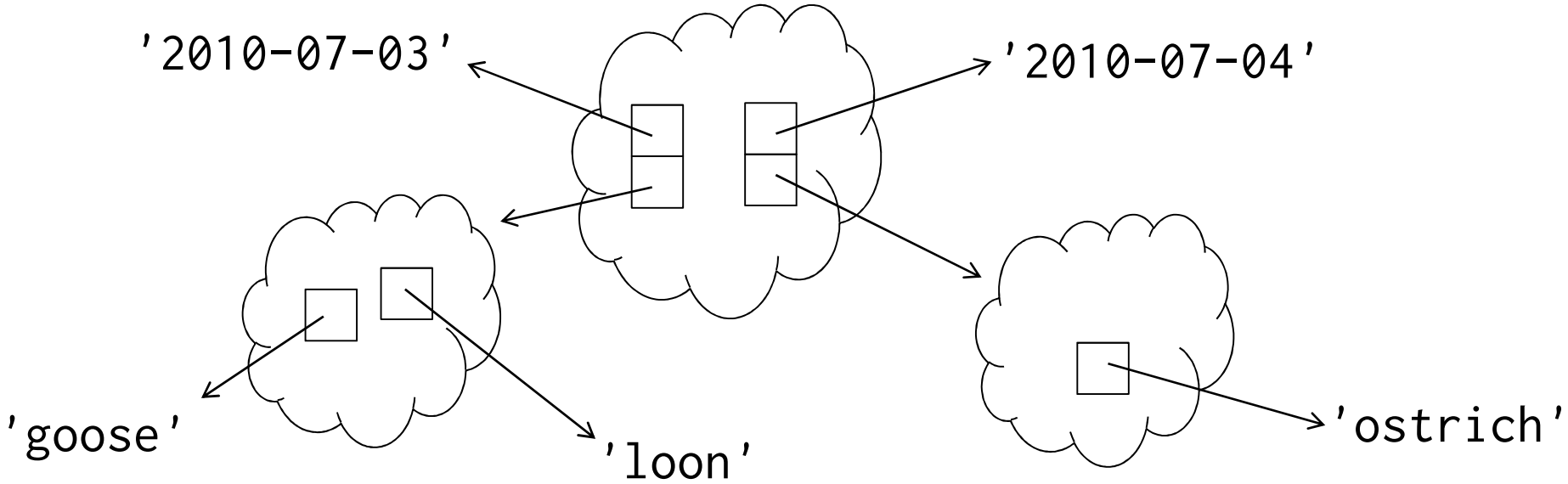
```
2010-07-03    05:38    loon
2010-07-03    06:02    goose
```



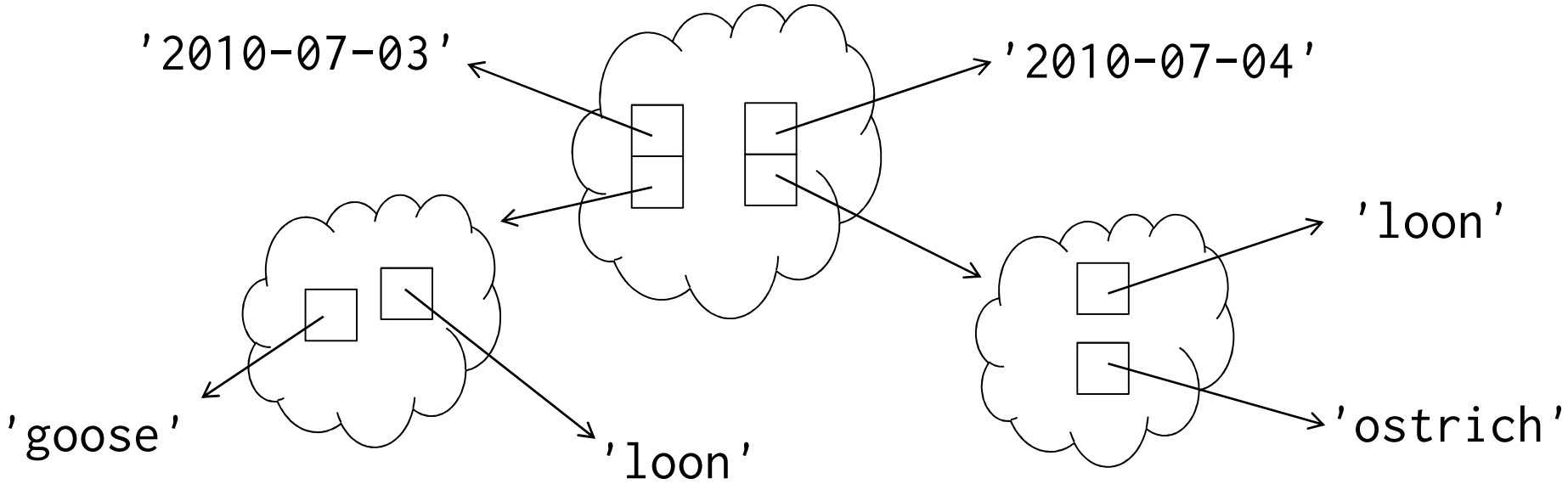
```
2010-07-03    05:38    loon
2010-07-03    06:02    goose
2010-07-03    06:07    loon
```



```
2010-07-03    05:38    loon
2010-07-03    06:02    goose
2010-07-03    06:07    loon
2010-07-04    05:09    ostrich   # hallucinating?
```



```
2010-07-03    05:38    loon
2010-07-03    06:02    goose
2010-07-03    06:07    loon
2010-07-04    05:09    ostrich   # hallucinating?
2010-07-04    05:29    loon
```



Which bird did we see least frequently?

Which bird did we see least frequently?

Actually, which *birds*, since two or more could be tied for the low score

Which bird did we see least frequently?

Actually, which *birds*, since two or more could be tied for the low score

Two-pass algorithm

Which bird did we see least frequently?

Actually, which *birds*, since two or more could be tied for the low score

Two-pass algorithm

- Find the minimum value in the dictionary

Which bird did we see least frequently?

Actually, which *birds*, since two or more could be tied for the low score

Two-pass algorithm

- Find the minimum value in the dictionary
- Find all keys with that value

Which bird did we see least frequently?

Actually, which *birds*, since two or more could be tied for the low score

Two-pass algorithm

- Find the minimum value in the dictionary
- Find all keys with that value

Combine these calculations in a one-pass algorithm

Which bird did we see least frequently?

Actually, which *birds*, since two or more could be tied for the low score

Two-pass algorithm

- Find the minimum value in the dictionary
- Find all keys with that value

Combine these calculations in a one-pass algorithm

Assume we already have a dictionary `counts` recording how often each kind of bird was seen

```
def least_frequently_seen(counts):
  '''Which bird(s) were least frequently seen?'''

  result = set()
  number = 0
  for bird in counts:
    …handle this bird…

  return result
```

```
def least_frequently_seen(counts):
  '''Which bird(s) were least frequently seen?'''

  result = set()
  number = 0
  for bird in counts:
    …handle this bird…

  return result
```

```
if len(result) == 0:
  result = {bird}
  number = counts[bird]
elif counts[bird] < number:
  result = {bird}
  number = counts[bird]
elif counts[bird] == number:
  result.add(bird)
```

```
def least_frequently_seen(counts):
  '''Which bird(s) were least frequently seen?'''

  result = set()
  number = 0
  for bird in counts:
    …handle this bird…

  return result
```

```
if len(result) == 0:
  result = {bird}
  number = counts[bird]
elif counts[bird] < number:
  result = {bird}
  number = counts[bird]
elif counts[bird] == number:
  result.add(bird)
```

Case 1: first bird (initializing data structures)

```
def least_frequently_seen(counts):
  '''Which bird(s) were least frequently seen?'''

  result = set()
  number = 0
  for bird in counts:
    …handle this bird…

  return result
```

```
if len(result) == 0:
  result = {bird}
  number = counts[bird]
elif counts[bird] < number:
  result = {bird}
  number = counts[bird]
elif counts[bird] == number:
  result.add(bird)
```

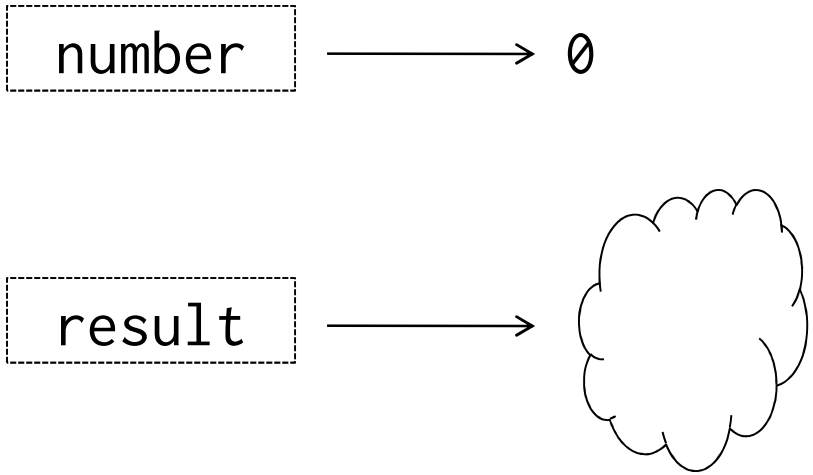
Case 2: new minimum, so replace everything

```
def least_frequently_seen(counts):
  '''Which bird(s) were least frequently seen?'''

  result = set()
  number = 0
  for bird in counts:
    …handle this bird…

  return result
```

```
if len(result) == 0:
  result = {bird}
  number = counts[bird]
elif counts[bird] < number:
  result = {bird}
  number = counts[bird]
elif counts[bird] == number:
  result.add(bird)
```

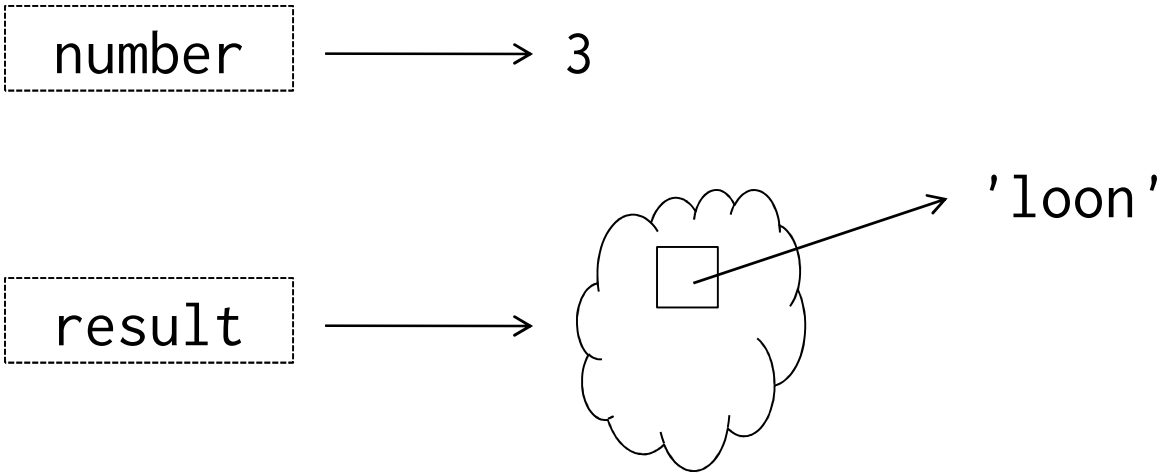
Case 3: tied equal for minimum

```
{ 'loon' : 3, 'goose' : 1, 'ostrich' : 1 }
```



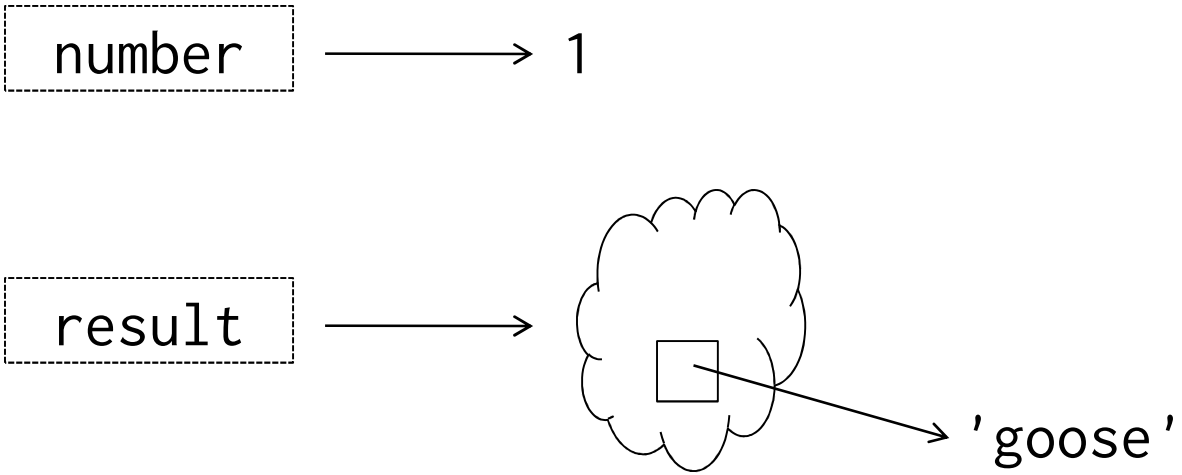
Before the loop

```
{ 'loon' : 3, 'goose' : 1, 'ostrich' : 1 }
```



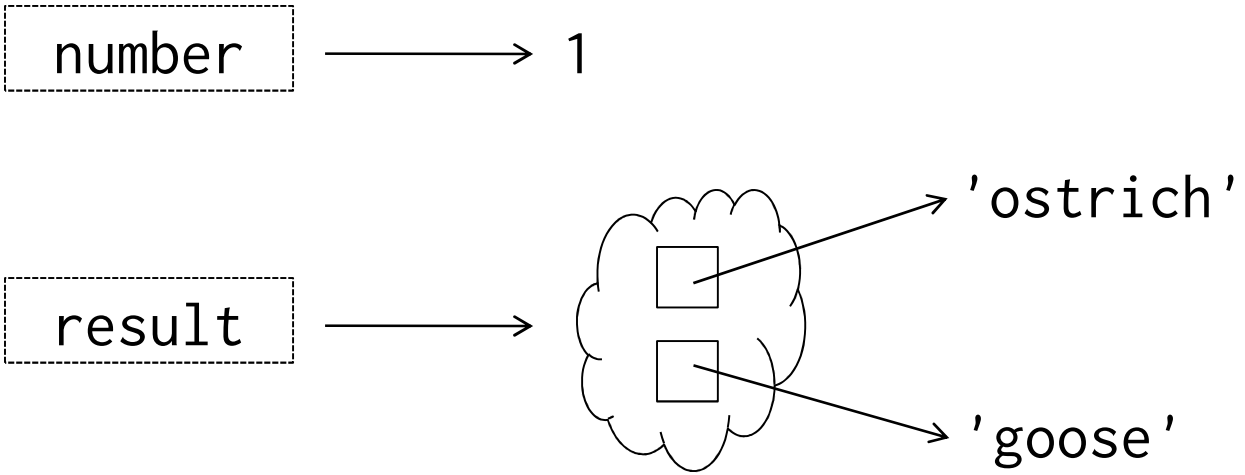
Case 1: first bird (initializing data structures)

```
{ 'loon' : 3, 'goose' : 1, 'ostrich' : 1 }
```



Case 2: new minimum, so replace everything

```
{ 'loon' : 3, 'goose' : 1, 'ostrich' : 1 }
```



Case 3: tied equal for minimum

software carpentry

created by

Greg Wilson

July 2010