Built-in Data Structures II: Tuples and Dictionaries

Contents

Tuples

- Create tuples
  - Items separated by commas

```
colors = 'red', 'blue', 'green'
mixed = ('Jack', 32.5, [1, 2])

print(type(colors))
print(type(mixed))
```

```
<class 'tuple'>
<class 'tuple'>
```

`tuple` type objects

Tuples

- Create tuples
  - Special cases

```
feel_empty = ()

print(type(feel_empty))
print(len(feel_empty))
```

```
<class 'tuple'>
0
```

Tuples

- Create tuples
  - Special cases

```
tuple_one = 'here',
item_one = ('there')

print(type(tuple_one))
print(type(item_one))
```

The comma is necessary in creating a tuple with one item

```
<class 'tuple'>
<class 'str'>
```

Tuples

- Create tuples
  - Special cases

```
tuple_one = 'here',
item_one = ('there')

print(type(tuple_one))
print(type(item_one))
```

It is the data item itself as the comma is missing

```
<class 'tuple'>
<class 'str'>
```

Tuples

- Basic features of tuples
  - Iterable
  - The same `len()` function, indexing and slicing expression as strings and lists
  - The same operations with `+` and `*` as strings and lists
  - Immutable
  - No method is defined for tuple objects

Tuples

- Basic features of tuples

```
letters = 'A', 'B', 'C'
numbers = 2.0, 3.0
```

```
mixed = letters + numbers*3
print(mixed)
print(len(mixed))
```

```
('A', 'B', 'C', 2.0, 3.0, 2.0, 3.0, 2.0, 3.0)
9
```

```
for item in mixed[6:]:
    print(item)
```

```
3.0
2.0
3.0
```

Tuples

- Basic features of tuples

```
letters = 'A', 'B', 'C'
numbers = 2.0, 3.0
```

```
mixed = letters + numbers*3
print(mixed)
print(len(mixed))
```

```
('A', 'B', 'C', 2.0, 3.0, 2.0, 3.0, 2.0, 3.0)
9
```

```
for item in mixed[6:]:
    print(item)
```

```
3.0
2.0
3.0
```

Tuples

- Tuple unpacking
  - Parallel assignment

```
location = 1.290, 103.852

latitude, longitude = location

print(latitude)
print(longitude)
```

```
1.29
103.852
```

```
x, y, z = 'abc'

print(x)
print(y)
print(z)
```

```
a
b
c
```

Tuples

- Tuple unpacking
  - Parallel assignment

```
location = (1.290, 103.852)

latitude, longitude = location

print(latitude)
print(longitude)
```

```
1.29
103.852
```

```
x, y, z = 'abc'

print(x)
print(y)
print(z)
```

```
a
b
c
```

Tuples

- Tuple unpacking
  - Parallel assignment

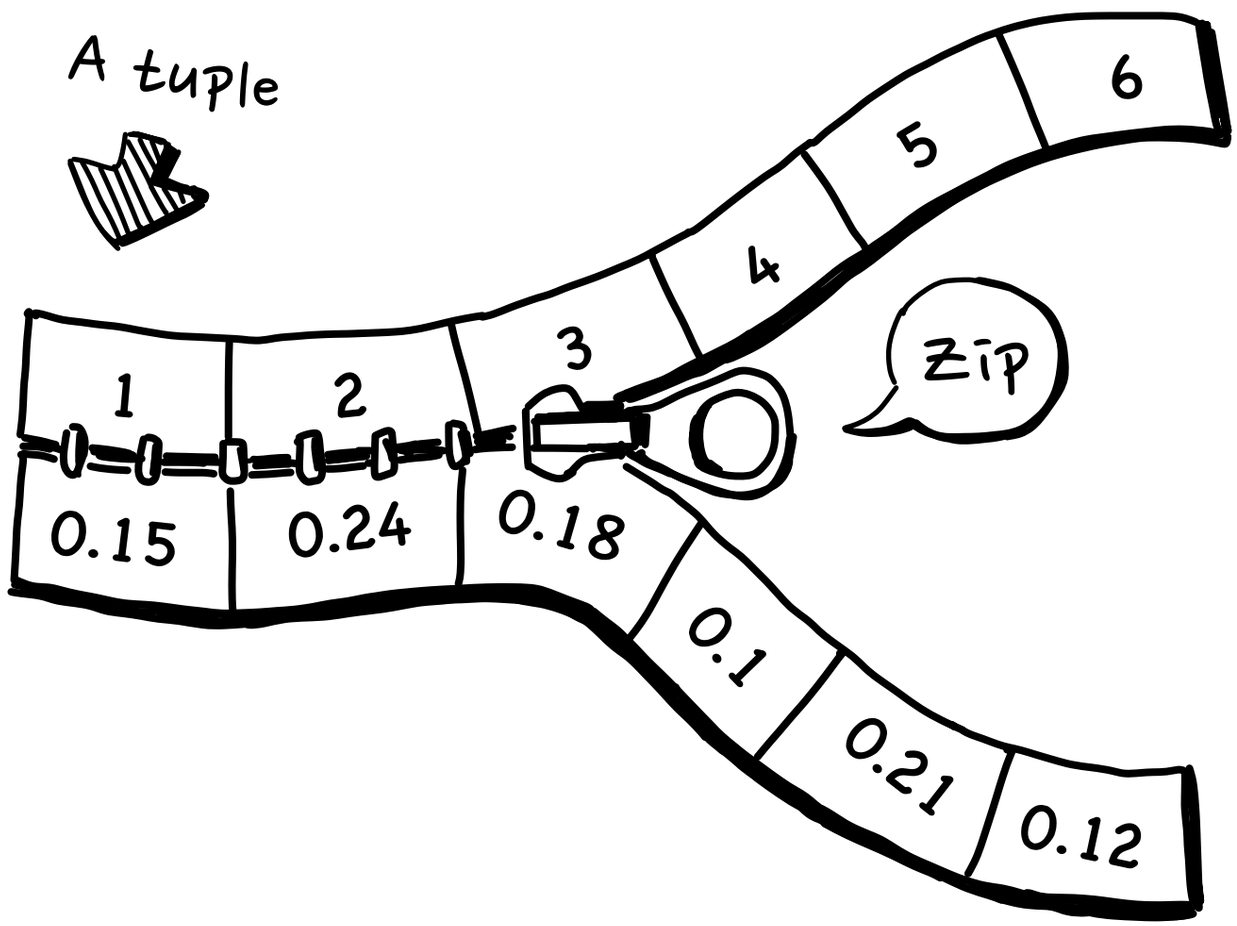
Example 1: There are two variables **`cage`** and **`travolta`** and their values are "bad guy" and "good guy", respectively. Swap the values of **`cage`** and **`travolta`** so that **`cage`** becomes "good guy" and **`travolta`** becomes "bad guy".

```
cage = 'bad guy'
travolta = 'good guy'

cage, travolta = travolta, cage

print(cage)
print(travolta)
```

```
good guy
bad guy
```

Tuples

- Tuple unpacking
  - Iteration in parallel with the `zip()` function



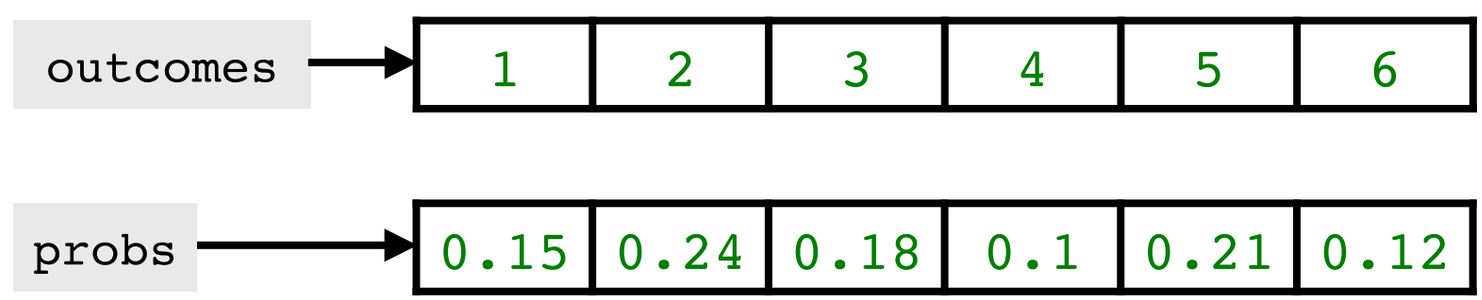
Tuples

- Tuple unpacking
  - Iteration in parallel with the `zip()` function

Example 2: For a biased die, each outcome of rolled number and the corresponding probability are given in lists **`outcomes`** and **`probs`**, respectively. Calculate the expected value of the rolled numbers.

```
outcomes = [1, 2, 3, 4, 5, 6]                    # Rolled numbers
probs = [0.15, 0.24, 0.18, 0.1, 0.21, 0.12]      # Probabilities
```

$$\mathbb{E}(X) = \sum_{i=1}^{6} x_i p_i$$



Tuples

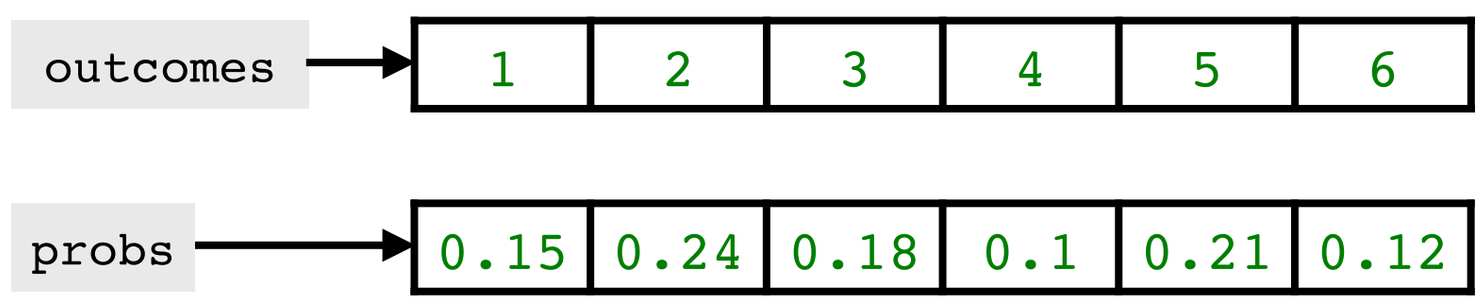
- Tuple unpacking
  - Iteration in parallel with the `zip()` function

```
for item in zip(outcomes, probs):
    print(item)
```

```
(1, 0.15)
(2, 0.24)
(3, 0.18)
(4, 0.1)
(5, 0.21)
(6, 0.12)
```

```
print(type(zip(outcomes, probs)))
```

```
<class 'zip'>
```

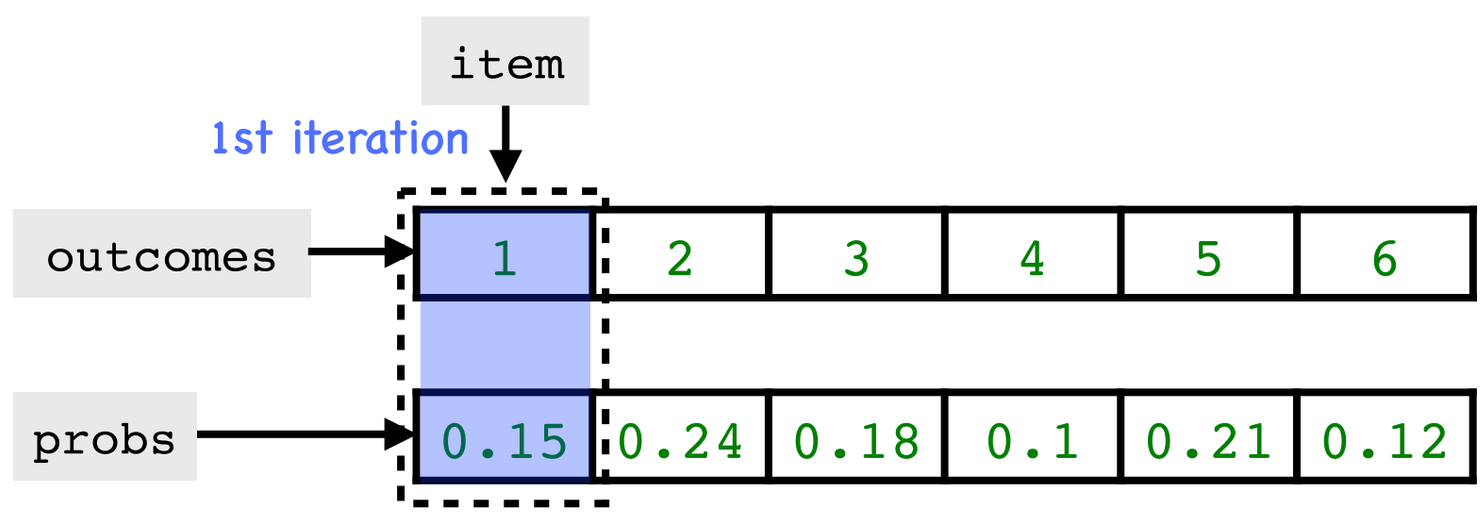


Tuples

- Tuple unpacking
  - Iteration in parallel with the `zip()` function

```
for item in zip(outcomes, probs):
    print(item)
```

```
(1, 0.15)
(2, 0.24)
(3, 0.18)
(4, 0.1)
(5, 0.21)
(6, 0.12)
```

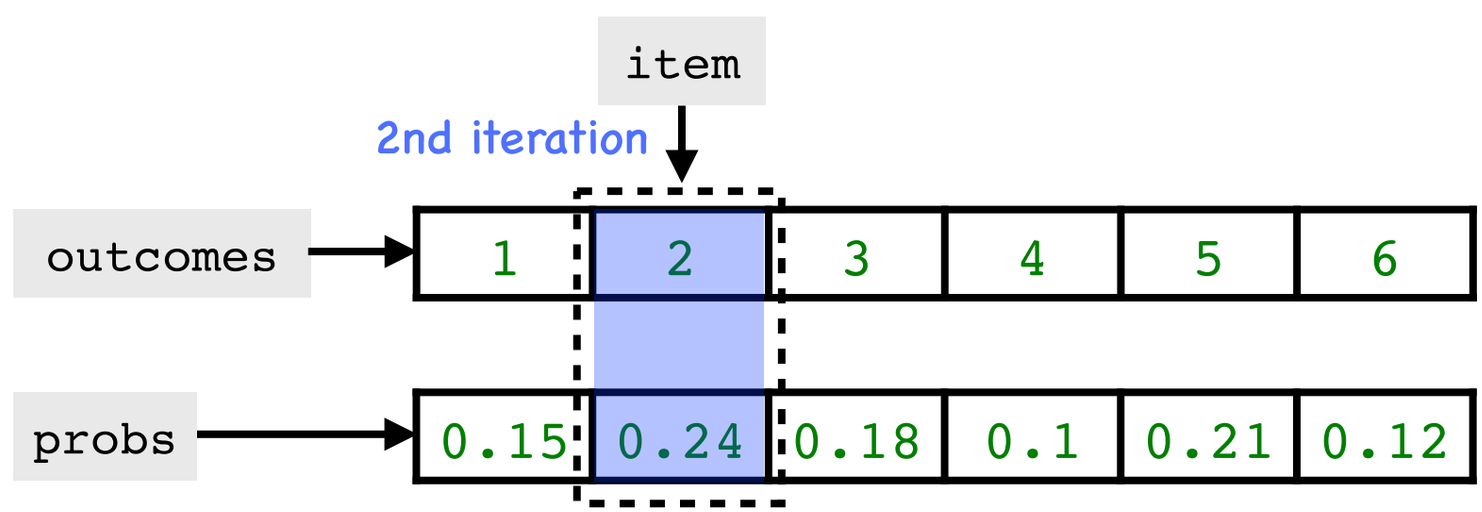


Tuples

- Tuple unpacking
  - Iteration in parallel with the `zip()` function

```
for item in zip(outcomes, probs):
    print(item)
```

```
(1, 0.15)
(2, 0.24)
(3, 0.18)
(4, 0.1)
(5, 0.21)
(6, 0.12)
```

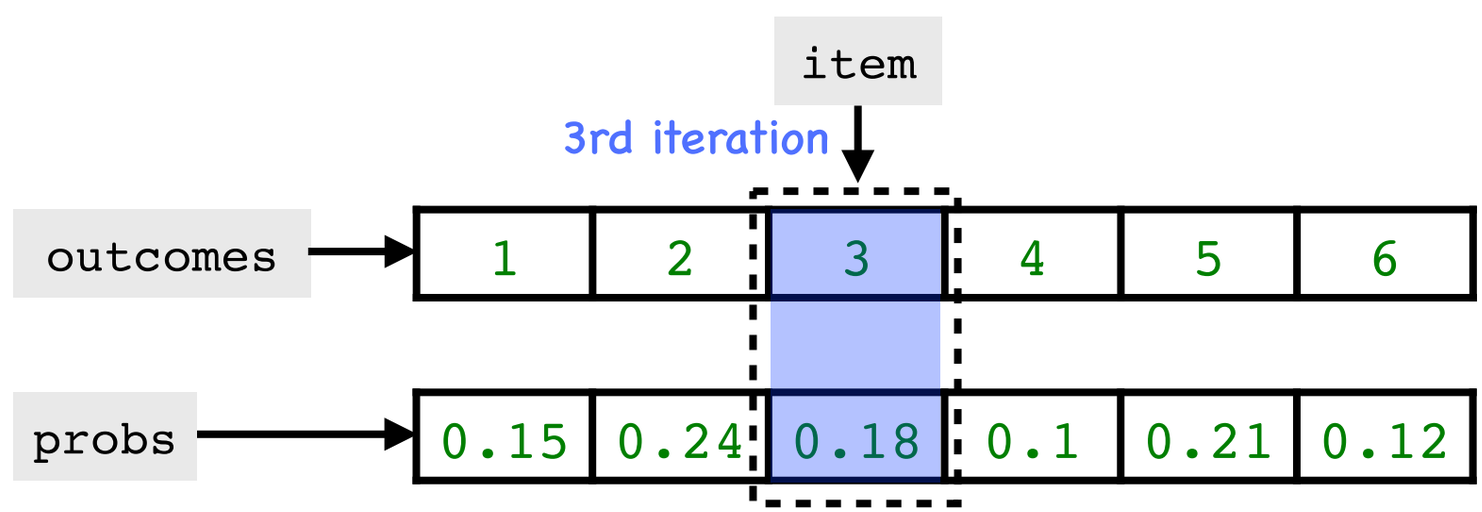


Tuples

- Tuple unpacking
  - Iteration in parallel with the `zip()` function

```
for item in zip(outcomes, probs):
    print(item)
```

```
(1, 0.15)
(2, 0.24)
(3, 0.18)
(4, 0.1)
(5, 0.21)
(6, 0.12)
```

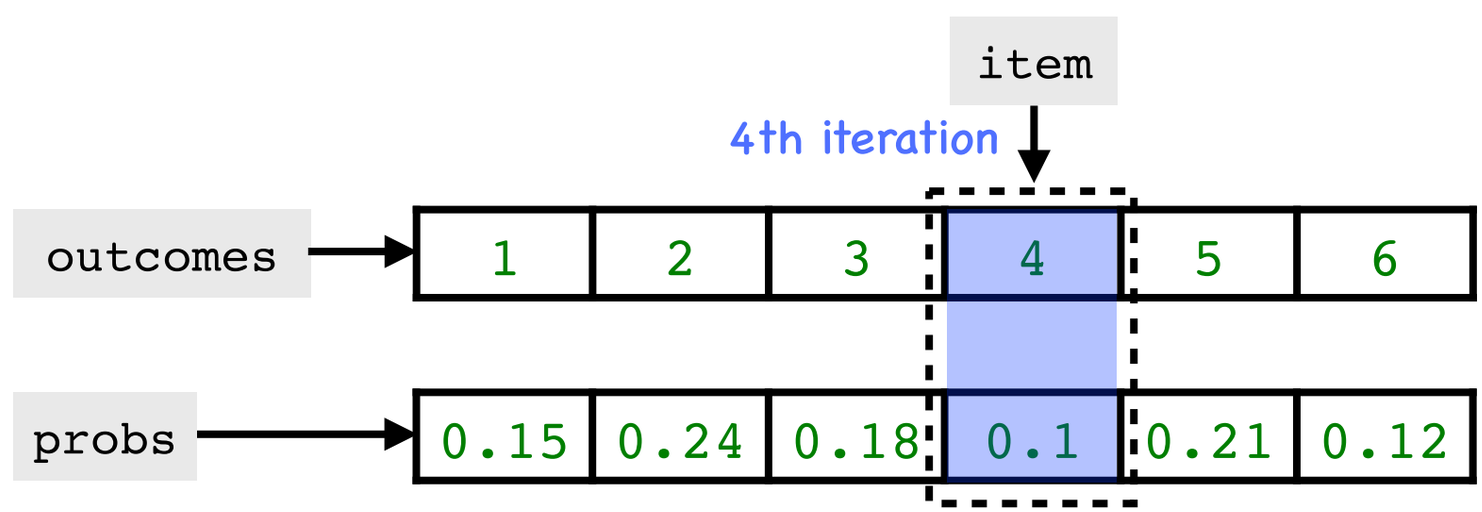


Tuples

- Tuple unpacking
  - Iteration in parallel with the `zip()` function

```
for item in zip(outcomes, probs):
    print(item)
```

```
(1, 0.15)
(2, 0.24)
(3, 0.18)
(4, 0.1)
(5, 0.21)
(6, 0.12)
```

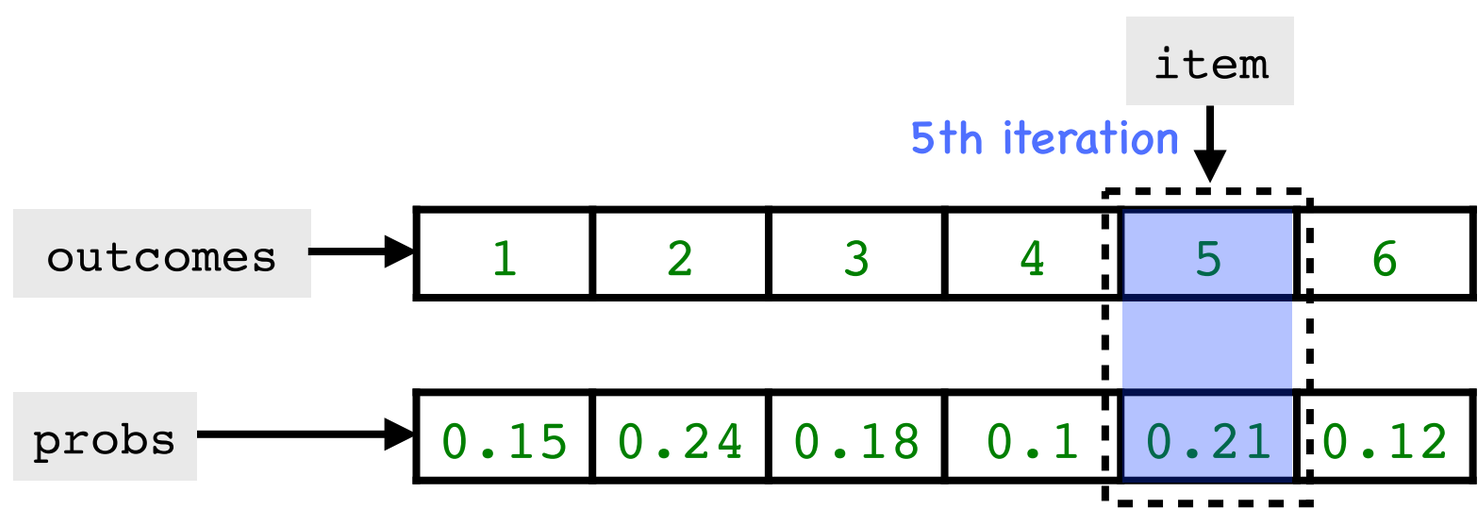


Tuples

- Tuple unpacking
  - Iteration in parallel with the `zip()` function

```
for item in zip(outcomes, probs):
    print(item)
```

```
(1, 0.15)
(2, 0.24)
(3, 0.18)
(4, 0.1)
(5, 0.21)
(6, 0.12)
```

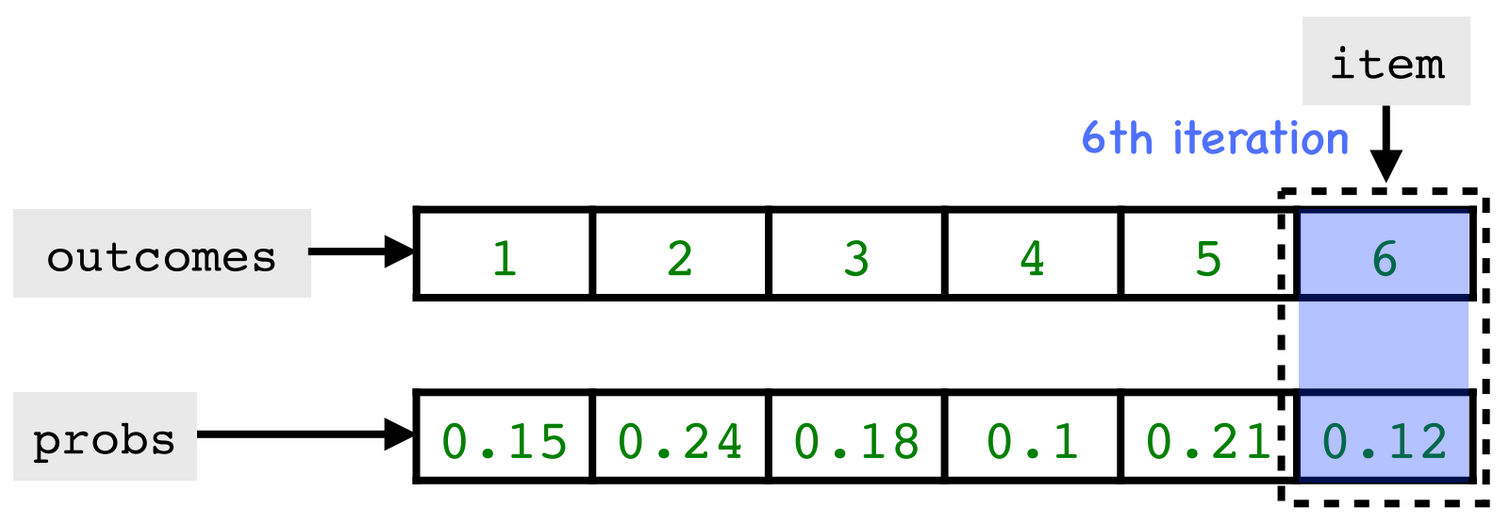


Tuples

- Tuple unpacking
  - Iteration in parallel with the `zip()` function

```
for item in zip(outcomes, probs):
    print(item)
```

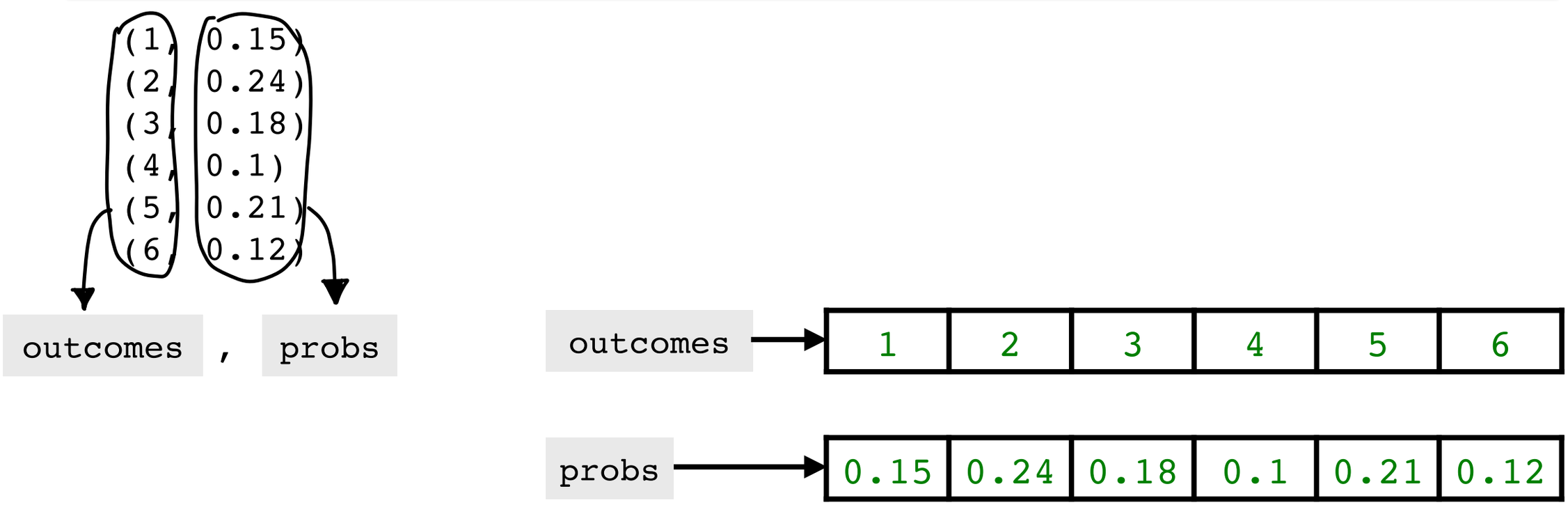
```
(1, 0.15)
(2, 0.24)
(3, 0.18)
(4, 0.1)
(5, 0.21)
(6, 0.12)
```



Tuples

- Tuple unpacking
  - Iteration in parallel with the `zip()` function

```
for item in zip(outcomes, probs):
    print(item)
```

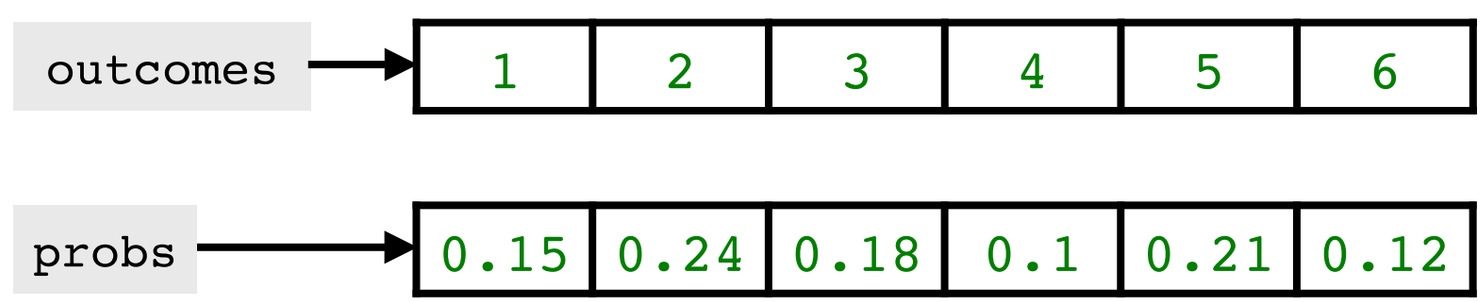


Tuples

- Tuple unpacking
  - Iteration in parallel with the `zip()` function

```
for outcome, prob in zip(outcomes, probs):
    print('outcome={}, prob={}'.format(outcome, prob))
```

```
outcome=1, prob=0.15
outcome=2, prob=0.24
outcome=3, prob=0.18
outcome=4, prob=0.1
outcome=5, prob=0.21
outcome=6, prob=0.12
```

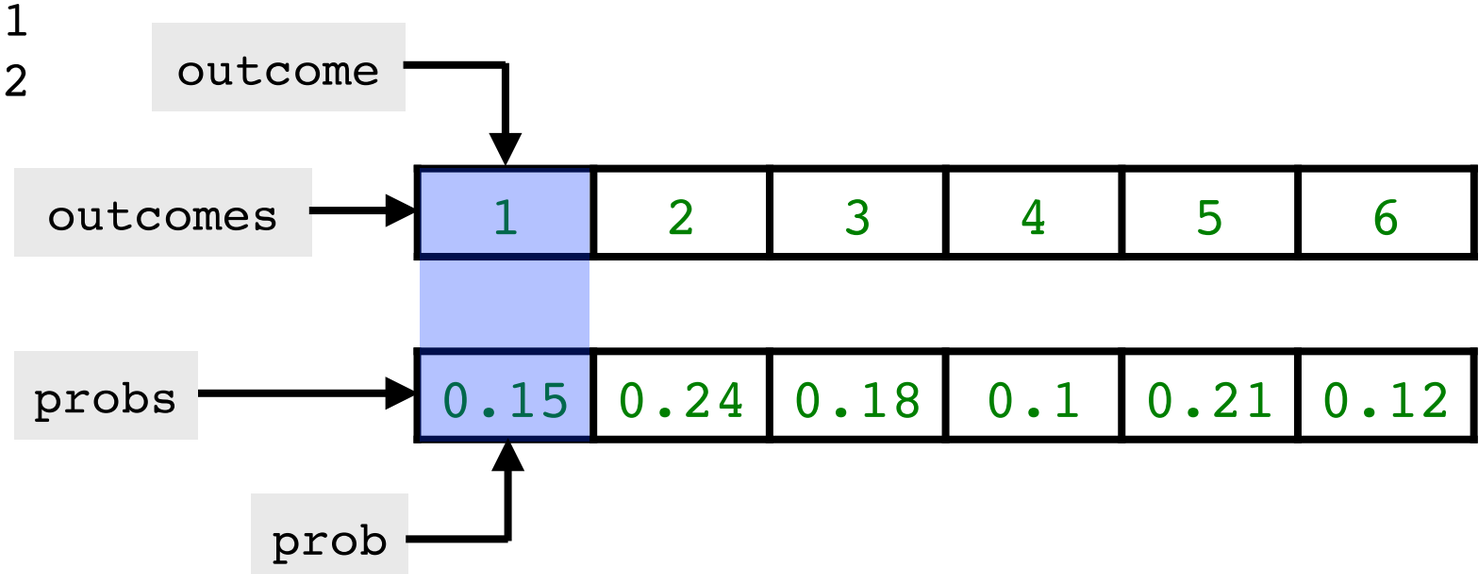


Tuples

- Tuple unpacking
  - Iteration in parallel with the `zip()` function

```
for outcome, prob in zip(outcomes, probs):
    print('outcome={}, prob={}'.format(outcome, prob))
```

```
outcome=1, prob=0.15
outcome=2, prob=0.24
outcome=3, prob=0.18
outcome=4, prob=0.1
outcome=5, prob=0.21
outcome=6, prob=0.12
```

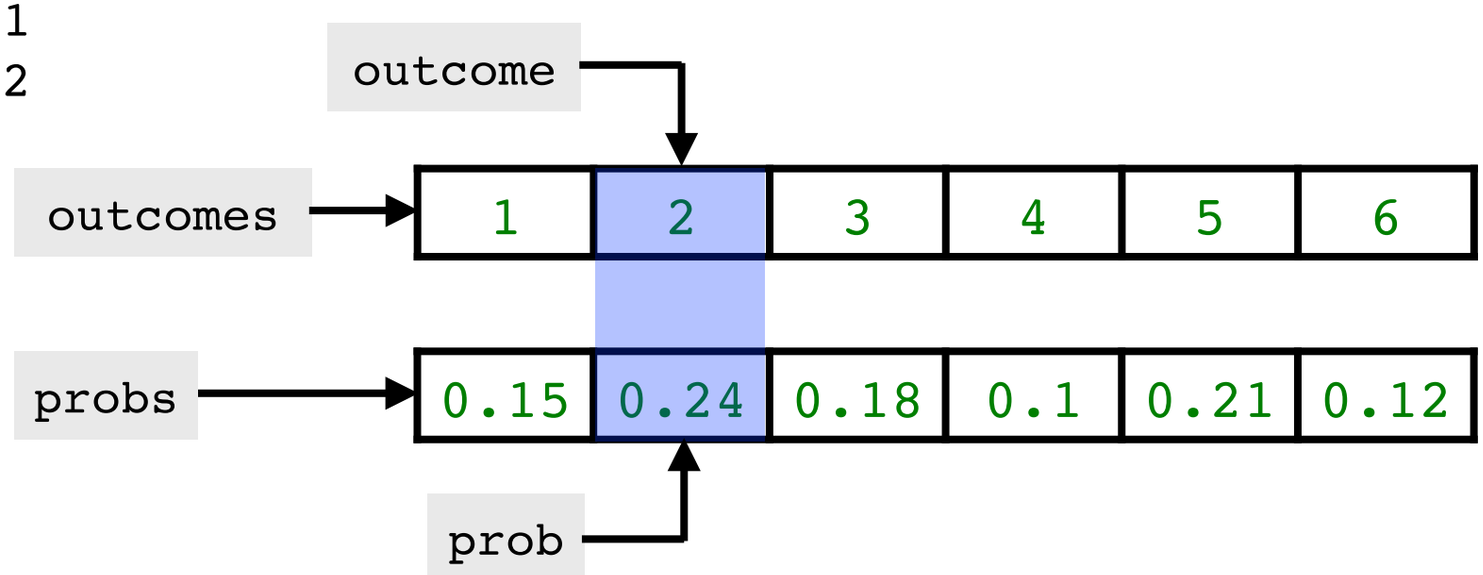


Tuples

- Tuple unpacking
  - Iteration in parallel with the `zip()` function

```
for outcome, prob in zip(outcomes, probs):
    print('outcome={}, prob={}'.format(outcome, prob))
```

```
outcome=1, prob=0.15
outcome=2, prob=0.24
outcome=3, prob=0.18
outcome=4, prob=0.1
outcome=5, prob=0.21
outcome=6, prob=0.12
```

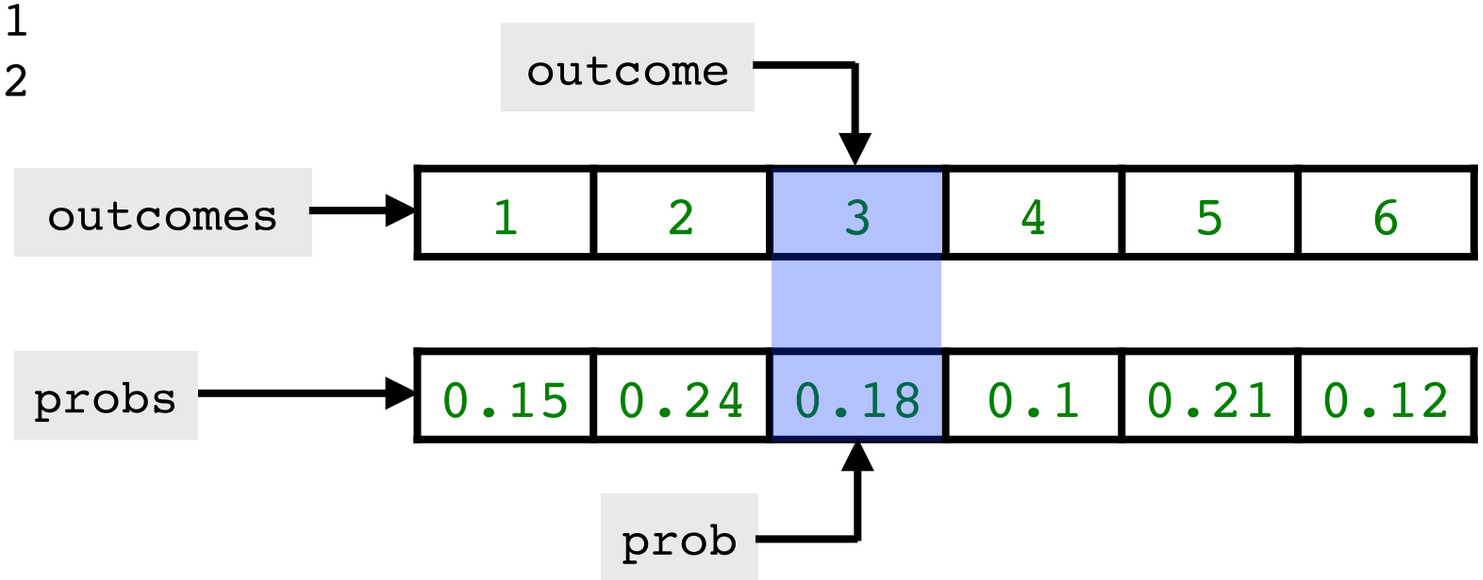


Tuples

- Tuple unpacking
  - Iteration in parallel with the `zip()` function

```
for outcome, prob in zip(outcomes, probs):
    print('outcome={}, prob={}'.format(outcome, prob))
```

```
outcome=1, prob=0.15
outcome=2, prob=0.24
outcome=3, prob=0.18
outcome=4, prob=0.1
outcome=5, prob=0.21
outcome=6, prob=0.12
```

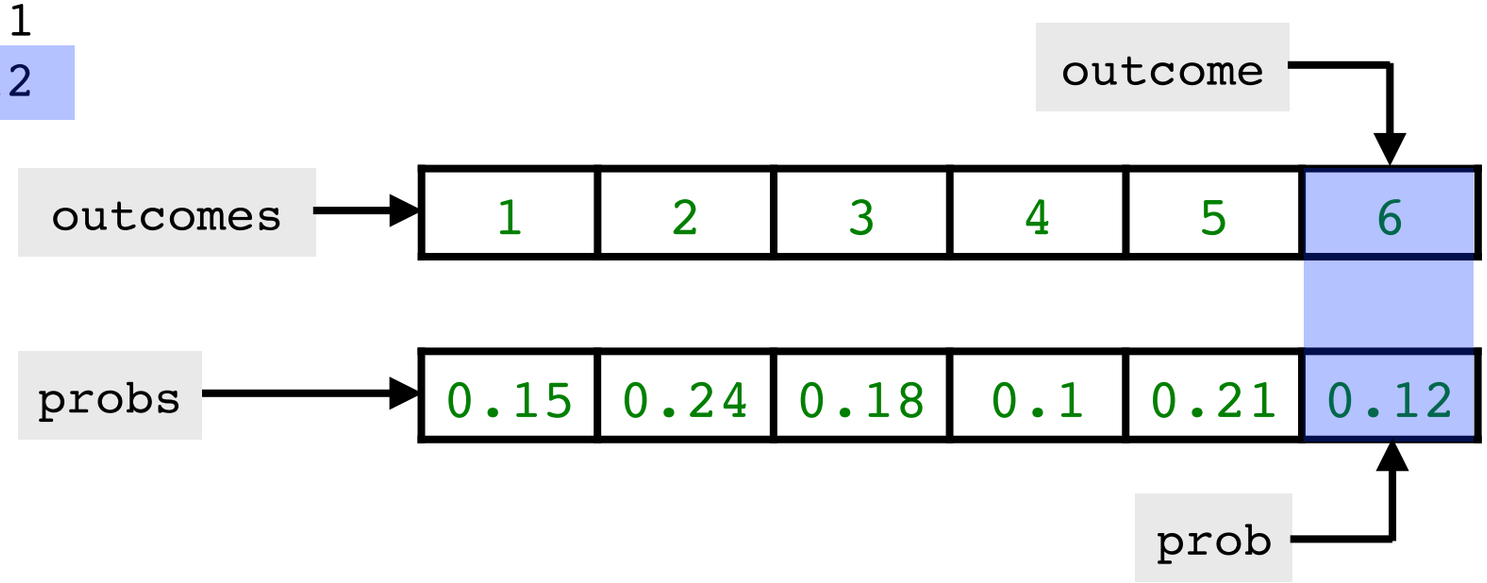


Tuples

- Tuple unpacking
  - Iteration in parallel with the `zip()` function

```
for outcome, prob in zip(outcomes, probs):
    print('outcome={}, prob={}'.format(outcome, prob))
```

```
outcome=1, prob=0.15
outcome=2, prob=0.24
outcome=3, prob=0.18
outcome=4, prob=0.1
outcome=5, prob=0.21
outcome=6, prob=0.12
```



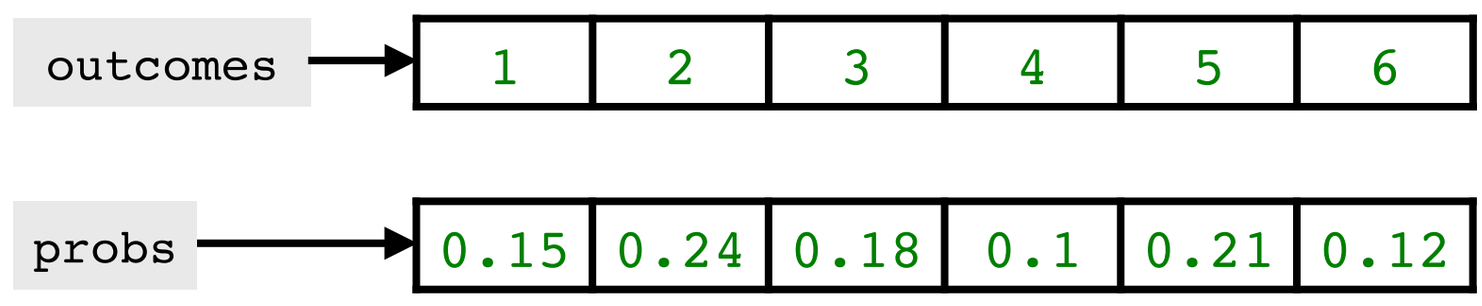
Tuples

- Tuple unpacking
  - Iteration in parallel with the `zip()` function

```
for outcome, prob in zip(outcomes, probs):
    print('outcome={}, prob={}'.format(outcome, prob))
```

```
outcome=1, prob=0.15
outcome=2, prob=0.24
outcome=3, prob=0.18
outcome=4, prob=0.1
outcome=5, prob=0.21
outcome=6, prob=0.12
```

$$\mathbb{E}(X) = \sum_{i=1}^{6} x_i p_i$$

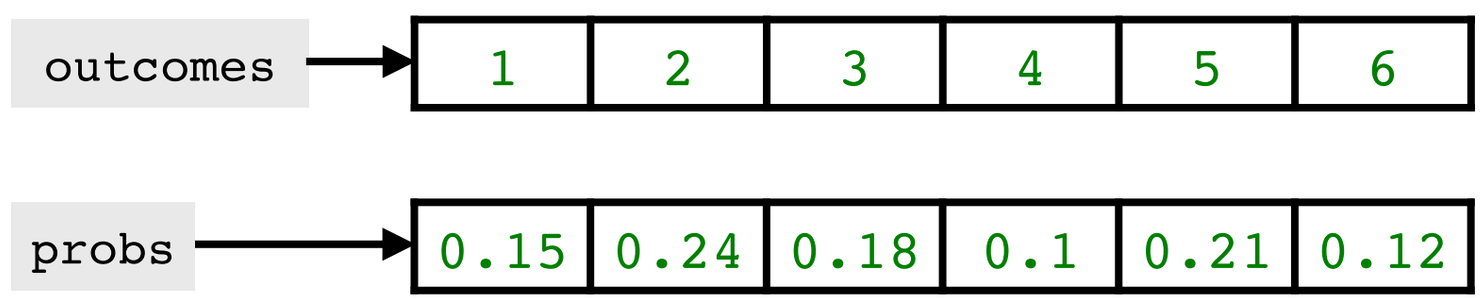


Tuples

- Tuple unpacking
  - Iteration in parallel with the `zip()` function

```
exp_value = 0
for ... ...:
    ... ...

print(exp_value)
```

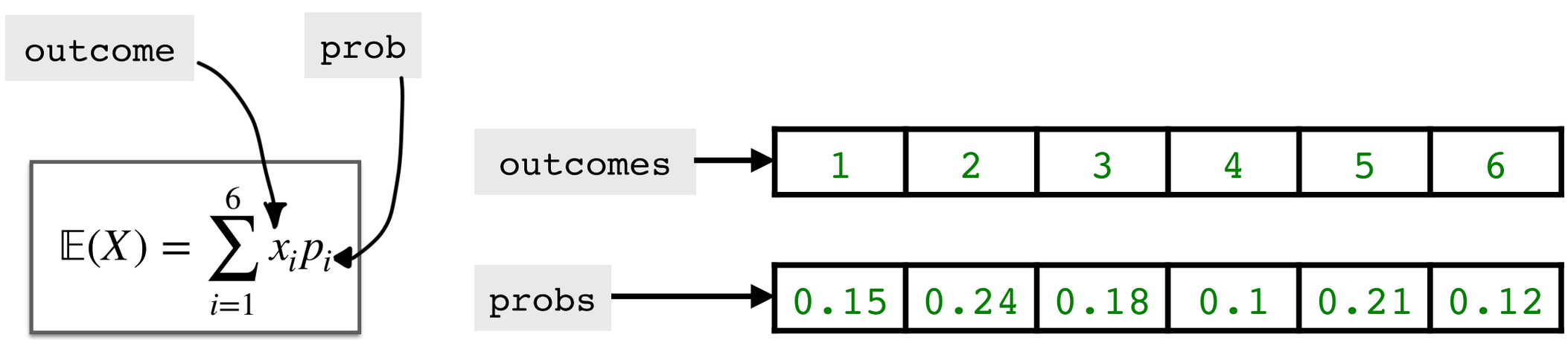
$$\mathbb{E}(X) = \sum_{i=1}^{6} x_i p_i$$



Tuples

- Tuple unpacking
  - Iteration in parallel with the `zip()` function

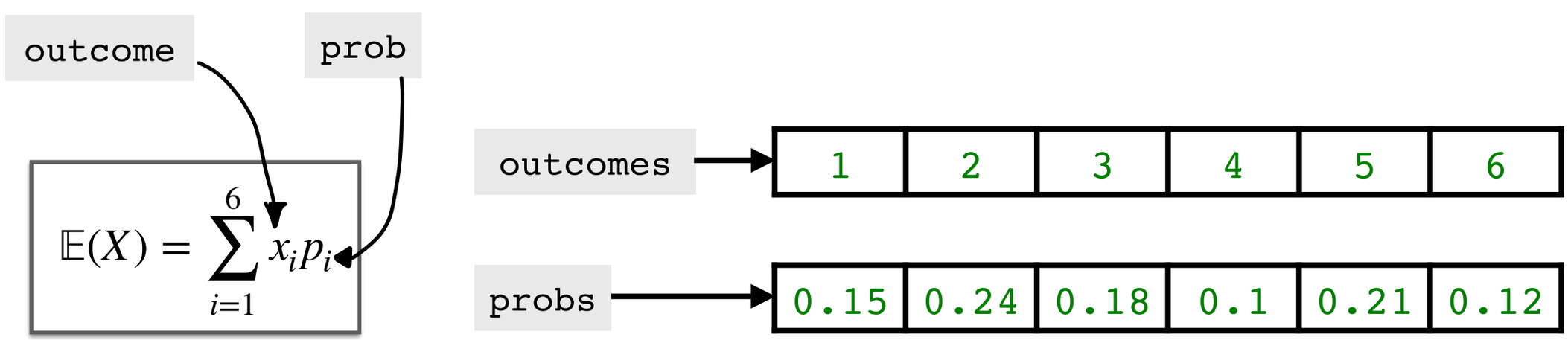
```
exp_value = 0
for outcome, prob in zip(outcomes, probs):
    ... ...

print(exp_value)
```



Tuples

- Tuple unpacking
  - Iteration in parallel with the `zip()` function

```
exp_value = 0
for outcome, prob in zip(outcomes, probs):
    exp_value += outcome*prob

print(exp_value)
```



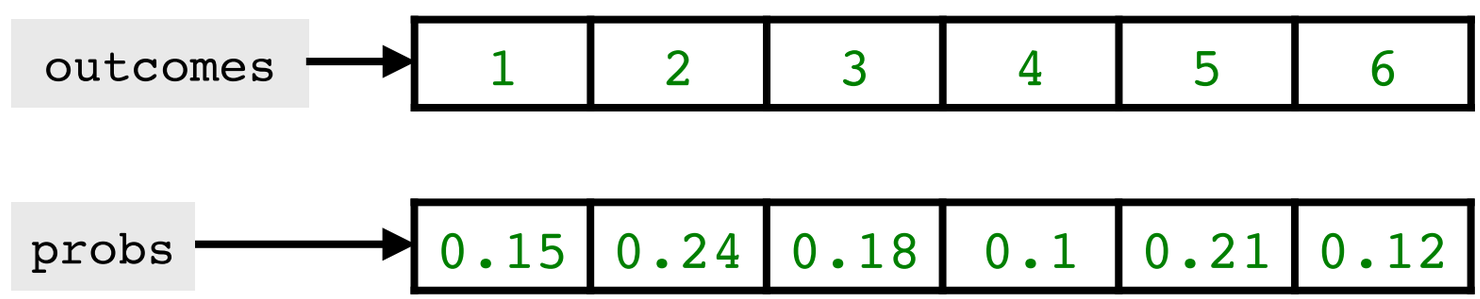
Tuples

- Tuple unpacking
  - Iteration in parallel with the `zip()` function

```
exp_value = 0
for outcome, prob in zip(outcomes, probs):
    exp_value += outcome*prob

print(exp_value)
```

```
3.34
```

$$\mathbb{E}(X) = \sum_{i=1}^{6} x_i p_i$$



Tuples

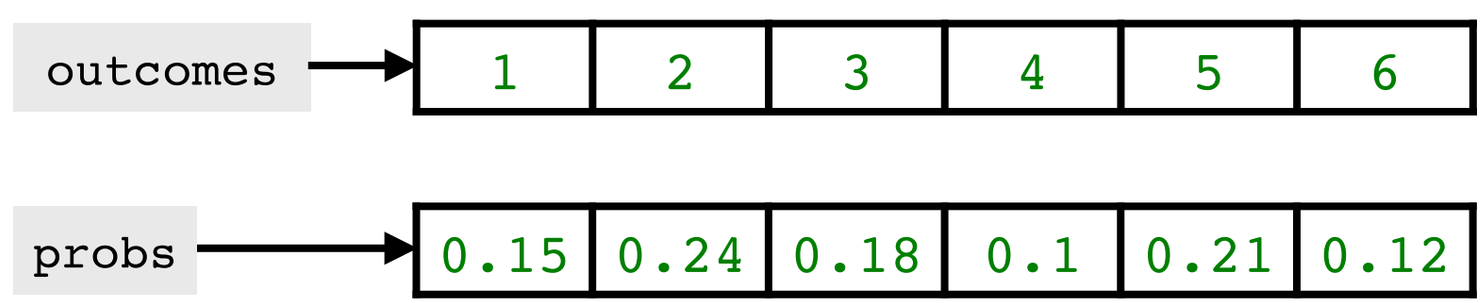
- Tuple unpacking
  - Iteration in parallel with the `zip()` function

```
[expression for item in iterable]
```

```
products = [outcome*prob
            for outcome, prob in zip(outcomes, probs)]
print(products)

exp_value = sum(products)
print(exp_value)
```

```
[0.15, 0.48, 0.54, 0.4, 1.05, 0.72]
3.34
```

$$\mathbb{E}(X) = \sum_{i=1}^{6} x_i p_i$$



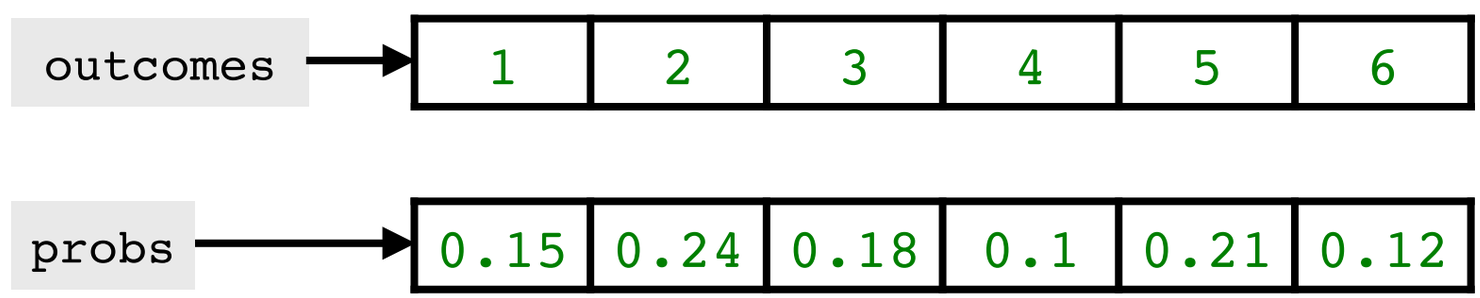
Tuples

- Tuple unpacking
  - Iteration in parallel with the `zip()` function

```
products = [outcome*prob
            for outcome, prob in zip(outcomes, probs)]
print(products)

exp_value = sum(products)
print(exp_value)
```

```
[0.15, 0.48, 0.54, 0.4, 1.05, 0.72]
3.34
```

$$\mathbb{E}(X) = \sum_{i=1}^{6} x_i p_i$$



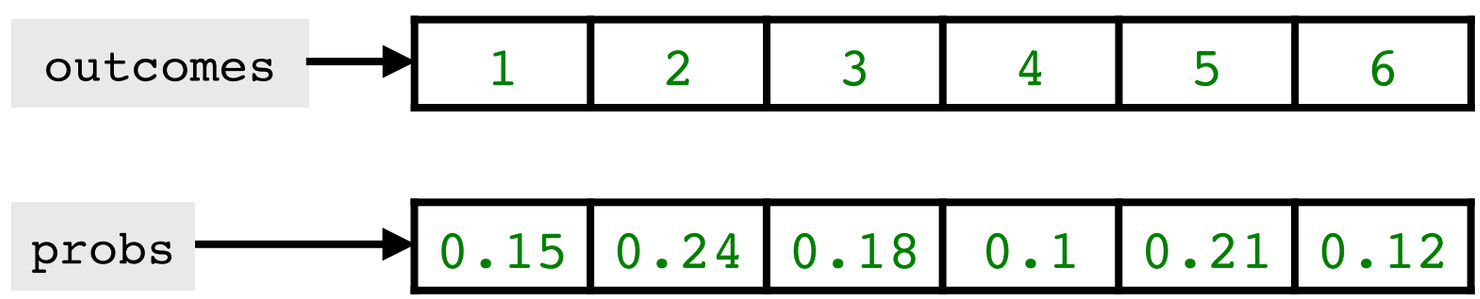
Tuples

- Tuple unpacking
  - Iteration in parallel with the `zip()` function

```
products = [outcome*prob
            for outcome, prob in zip(outcomes, probs)]
print(products)

exp_value = sum(products)
print(exp_value)
```

```
[0.15, 0.48, 0.54, 0.4, 1.05, 0.72]
3.34
```

$$\mathbb{E}(X) = \sum_{i=1}^{6} x_i p_i$$



Tuples

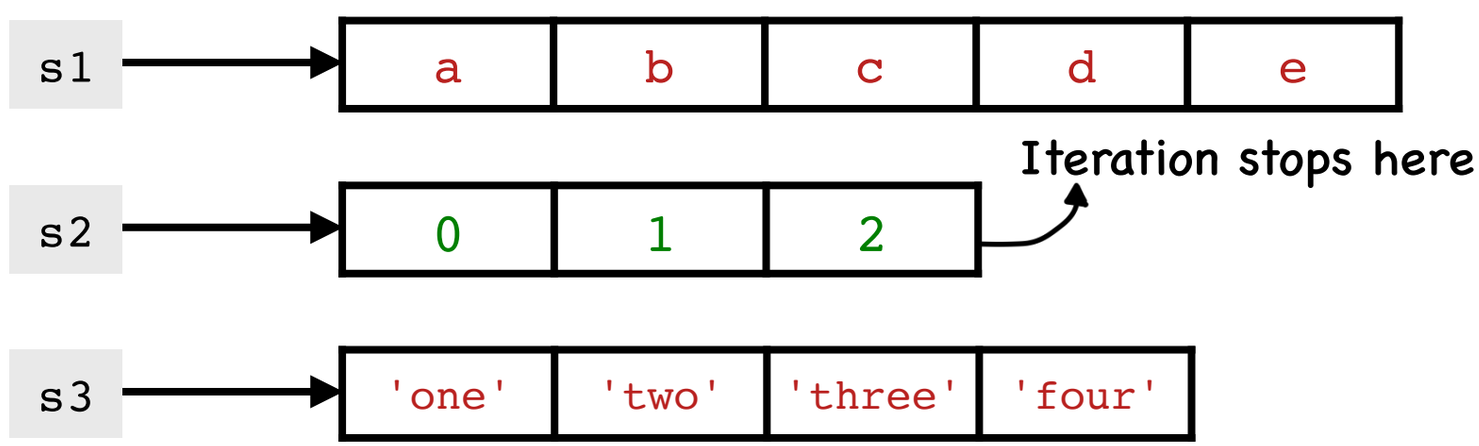
- Tuple unpacking
  - Iteration in parallel with the `zip()` function
    - ✓ The function can be applied to other iterable types
    - ✓ The function can be used to more than two data sequences
    - ✓ For sequences with different lengths, the iteration stops when the shortest sequence is running out of items

Tuples

- Tuple unpacking
  - Iteration in parallel with the `zip()` function

```
s1 = 'abcde'
s2 = range(3)
s3 = 'one', 'two', 'three', 'four'

for x, y, z in zip(s1, s2, s3):
    print(x, y, z)
```

```
a 0 one
b 1 two
c 2 three
```



Dictionaries

- Create dictionaries
  - Comma-separated `key:value` pairs enclosed in curly brackets

```
stocks = {'AMZN': 170.40,
          'TSLA': 130.11,
          'TWTR': 32.48,
          'AAPL': 76.60,
          'ORCL': 51.58,
          'GOOG': 1434.23}
```

```
print(type(stocks))
```

```
<class 'dict'>
```

dict for dictionaries

```
{'AMZN': 170.40,
 'TSLA': 130.11,
 'TWTR': 32.48,
 'AAPL': 76.6,
 'ORCL': 51.58,
 'GOOG': 1434.23}
```

Values	170.40	130.11	32.48	76.60	51.58	1434.23
Keys	'AMZN'	'TSLA'	'TWTR'	'AAPL'	'ORCL'	'GOOG'

Dictionaries

- Create dictionaries
  - Comma-separated `key:value` pairs enclosed in curly brackets
    - ✓ Dictionary keys can be any immutable types: boolean, integer, float, string, tuple, and others
    - ✓ Dictionary values can be any data types

Values →	170.40	130.11	32.48	76.60	51.58	1434.23
Keys →	'AMZN'	'TSLA'	'TWTR'	'AAPL'	'ORCL'	'GOOG'

Dictionaries

- Data arrangement of dictionaries
  - Comparison between lists and dictionaries

```
[170.40, 130.11, 32.48, 76.6, 51.58, 1434.23]
```

Values	170.40	130.11	32.48	76.60	51.58	1434.23
Indexes	0	1	2	3	4	5
Indexes from the rear	-6	-5	-4	-3	-2	-1

```
{'AMZN': 170.40,
 'TSLA': 130.11,
 'TWTR': 32.48,
 'AAPL': 76.6,
 'ORCL': 51.58,
 'GOOG': 1434.23}
```

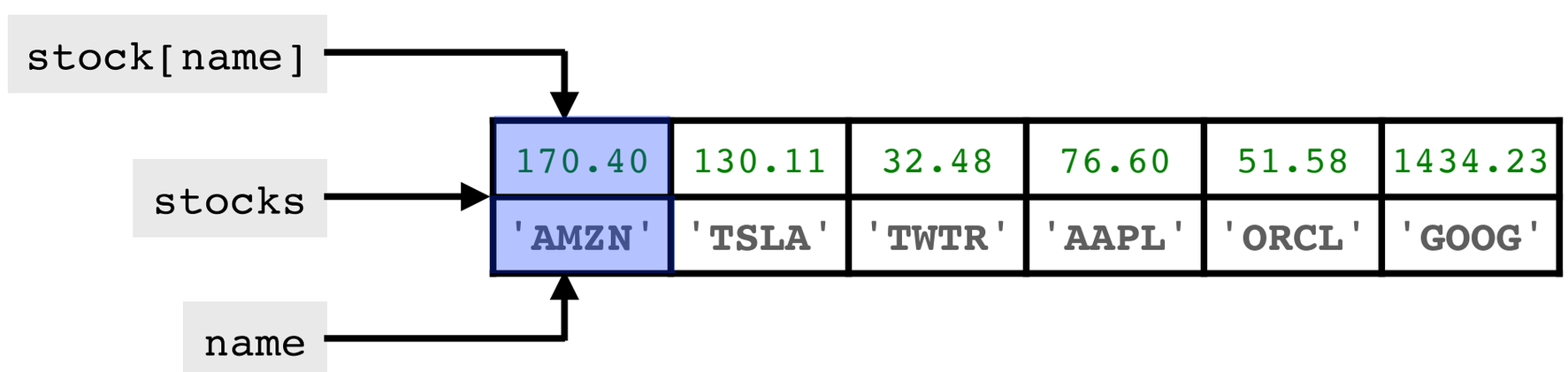
Values	170.40	130.11	32.48	76.60	51.58	1434.23
Keys	'AMZN'	'TSLA'	'TWTR'	'AAPL'	'ORCL'	'GOOG'

Dictionaries

- Data arrangement of dictionaries
  - Comparison between lists and dictionaries
    - ✓ Accessing items via keys

```
name = 'AMZN'
print('{}: {}'.format(name, stocks[name]))
```

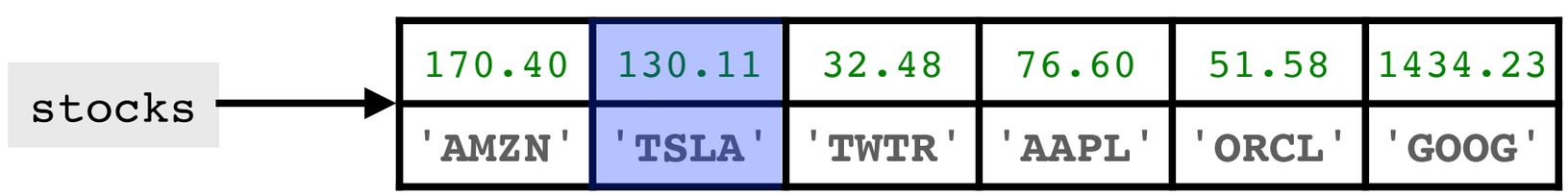
```
AMZN: 170.40
```



Dictionaries

- Data arrangement of dictionaries
  - Comparison between lists and dictionaries
    - ✓ Changing items in-place via keys

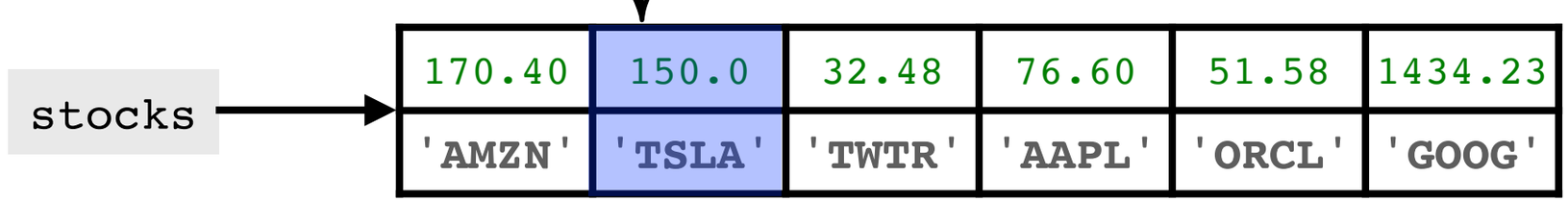
```
stocks['TSLA'] = 150.00 # Change the value of 'TSLA' to be 150.00
stocks['AAPL'] += 0.50  # Increase the value of 'AAPL' by 0.50
```



Dictionaries

- Data arrangement of dictionaries
  - Comparison between lists and dictionaries
    - ✓ Changing items in-place via keys

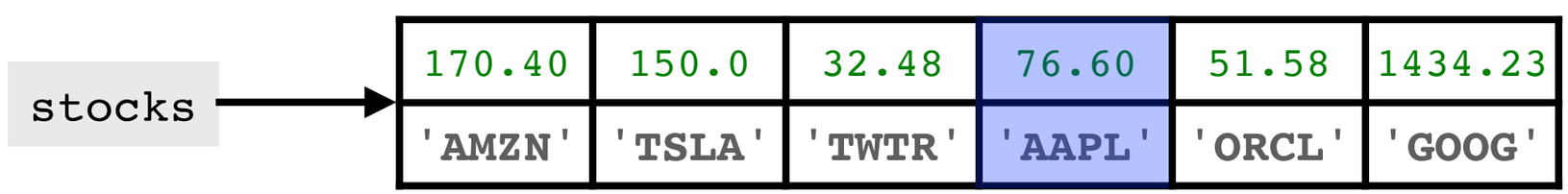
```
stocks['TSLA'] = 150.00 # Change the value of 'TSLA' to be 150.00
stocks['AAPL'] += 0.50  # Increase the value of 'AAPL' by 0.50
```



Dictionaries

- Data arrangement of dictionaries
  - Comparison between lists and dictionaries
    - ✓ Changing items in-place via keys

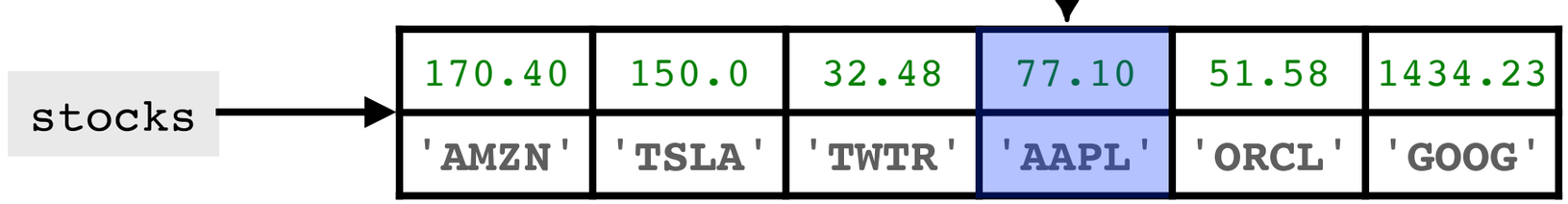
```
stocks['TSLA'] = 150.00 # Change the value of 'TSLA' to be 150.00
stocks['AAPL'] += 0.50  # Increase the value of 'AAPL' by 0.50
```



Dictionaries

- Data arrangement of dictionaries
  - Comparison between lists and dictionaries
    - ✓ Changing items in-place via keys

```
stocks['TSLA'] = 150.00 # Change the value of 'TSLA' to be 150.00
stocks['AAPL'] += 0.50  # Increase the value of 'AAPL' by 0.50
```



Dictionaries

- Data arrangement of dictionaries
  - Comparison between lists and dictionaries
    - ✓ Add new items as key-value pairs

```
stocks['ZM'] = 76.30     # Add a new key ZM and new value 76.30
```

A new key that is different from all existing keys

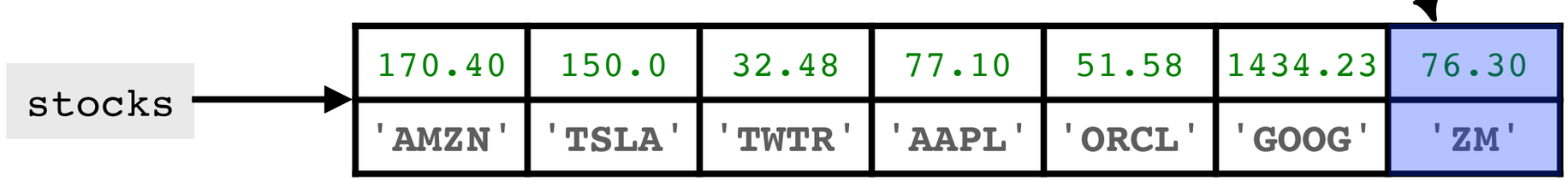
stocks →

170.40	150.0	32.48	77.10	51.58	1434.23
'AMZN'	'TSLA'	'TWTR'	'AAPL'	'ORCL'	'GOOG'

Dictionaries

- Data arrangement of dictionaries
  - Comparison between lists and dictionaries
    - ✓ Add new items as key-value pairs

```
stocks['ZM'] = 76.30    # Add a new key ZM and new value 76.30
```



Dictionaries

- Data arrangement of dictionaries
  - Comparison between lists and dictionaries

Example 3: The variable **`words`** is a list containing words of a song. Create a dictionary where the keys are all words appearing in the song, and the values are the numbers of appearances of these words.

Dictionaries

- Data arrangement of dictionaries
  - Comparison between lists and dictionaries

```
words = ['hey', 'jude', "don't", 'make', 'it', 'bad',
         'take', 'a', 'sad', 'song', 'and', 'make', 'it', 'better',
         'remember', 'to', 'let', 'her', 'into', 'your', 'heart',
         'then', 'you', 'can', 'start', 'to', 'make', 'it', 'better',
         'hey', 'jude', "don't", 'be', 'afraid',
         'you', 'were', 'made', 'to', 'go', 'out', 'and', 'get', 'her',
         'the', 'minute', 'you', 'let', 'her', 'under', 'your', 'skin',
         'then', 'you', 'begin', 'to', 'make', 'it', 'better']
```

Dictionaries

- Data arrangement of dictionaries
  - Comparison between lists and dictionaries

```
records = {}
for ... ...:
    ...
    ...
    ...
    ...

print(records)
```

Dictionaries

- Data arrangement of dictionaries
  - Comparison between lists and dictionaries

```
records = {}
for word in words:
    ...
    ...
    ...
    ...

print(records)
```

Dictionaries

- Data arrangement of dictionaries
  - Comparison between lists and dictionaries

```
records = {}
for word in words:
    if word in records:
        records[word] += 1
    else:
        records[word] = 1

print(records)
```

Dictionaries

- Data arrangement of dictionaries
  - Comparison between lists and dictionaries

```
records = {}
for word in words:
    if word in records:
        records[word] += 1
    else:
        records[word] = 1

print(records)
```

Increase the word counts by one as one more appearance is observed

Set the word counts to be one as it is the first time to observe the word

Dictionaries

- Basic features of dictionaries
  - Iterable
  - The same `len()` function
  - Indexing with keys
  - Mutable

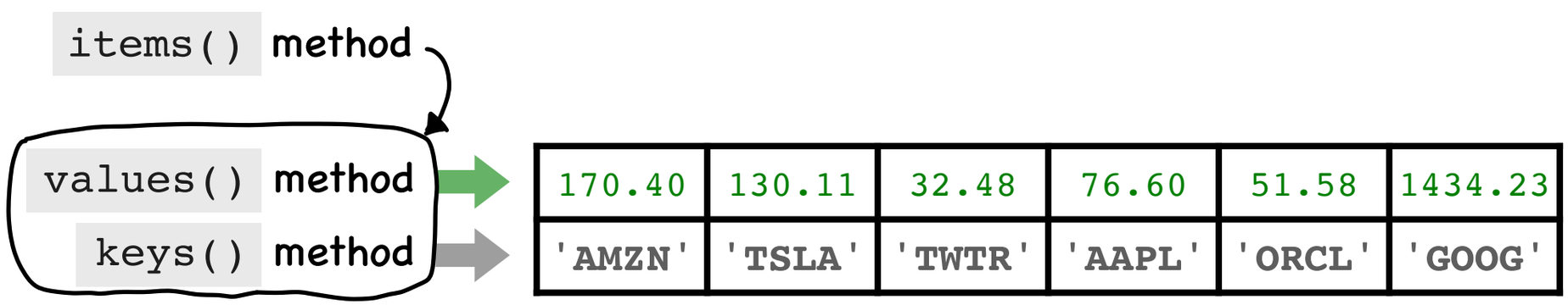
Dictionaries

- Loops and iterations with dictionaries
  - Iterating over keys
  - Iterating over values
  - Iterating over key-value items

Values →	170.40	130.11	32.48	76.60	51.58	1434.23
Keys →	'AMZN'	'TSLA'	'TWTR'	'AAPL'	'ORCL'	'GOOG'

Dictionaries

- Loops and iterations with dictionaries
  - Iterating over keys
  - Iterating over values
  - Iterating over key-value items



Dictionaries

- Loops and iterations with dictionaries
  - Iterating over keys

```
for name in stocks.keys():
    print(name)
```

```
AMZN
TSLA
TWTR
AAPL
ORCL
GOOG
```

`keys()` method

170.40	130.11	32.48	76.60	51.58	1434.23
'AMZN'	'TSLA'	'TWTR'	'AAPL'	'ORCL'	'GOOG'

Dictionaries

- Loops and iterations with dictionaries
  - Iterating over keys

```
for name in stocks.keys():
    print(name)
```

```
AMZN
TSLA
TWTR
AAPL
ORCL
GOOG
```

```
for name in stocks:
    print(name)
```

```
AMZN
TSLA
TWTR
AAPL
ORCL
GOOG
```

`keys()` method

170.40	130.11	32.48	76.60	51.58	1434.23
'AMZN'	'TSLA'	'TWTR'	'AAPL'	'ORCL'	'GOOG'

Dictionaries

- Loops and iterations with dictionaries
  - Iterating over keys

```
for name in stocks:
    print('{}: {}'.format(name, stocks[name]))
```

```
AMZN: 170.40
TSLA: 130.11
TWTR: 32.48
AAPL: 76.6
ORCL: 51.58
GOOG: 1434.23
```

Access the value of the data item via the key

`keys()` method

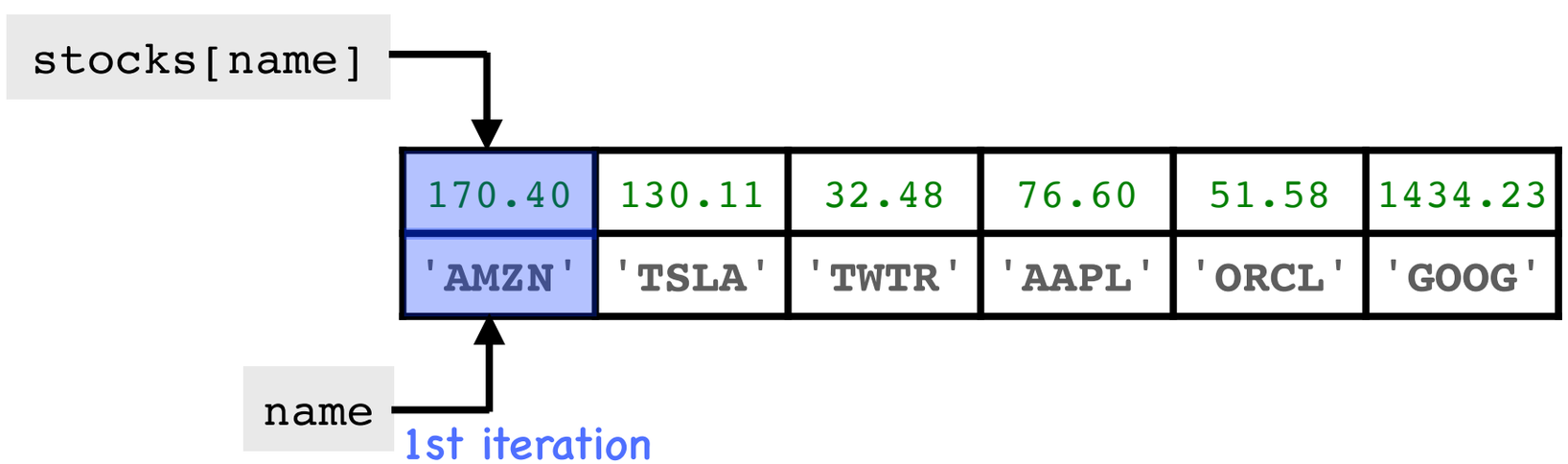
170.40	130.11	32.48	76.60	51.58	1434.23
'AMZN'	'TSLA'	'TWTR'	'AAPL'	'ORCL'	'GOOG'

Dictionaries

- Loops and iterations with dictionaries
  - Iterating over keys

```
for name in stocks:
    print('{}: {}'.format(name, stocks[name]))
```

```
AMZN: 170.40
TSLA: 130.11
TWTR: 32.48
AAPL: 76.6
ORCL: 51.58
GOOG: 1434.23
```

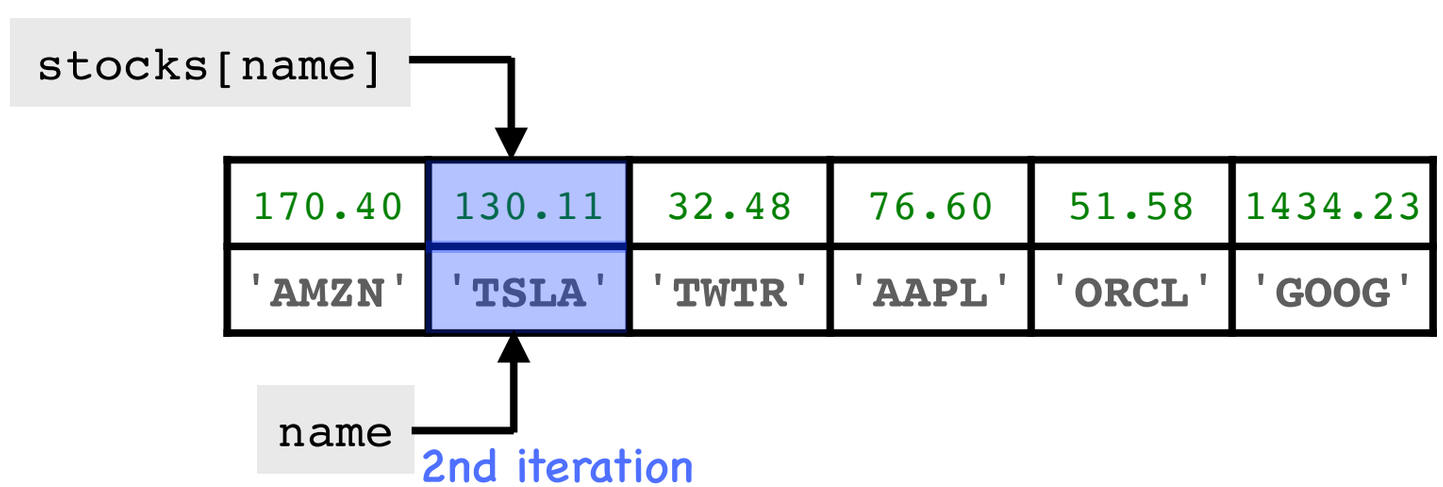


Dictionaries

- Loops and iterations with dictionaries
  - Iterating over keys

```
for name in stocks:
    print('{}: {}'.format(name, stocks[name]))
```

```
AMZN: 170.40
TSLA: 130.11
TWTR: 32.48
AAPL: 76.6
ORCL: 51.58
GOOG: 1434.23
```

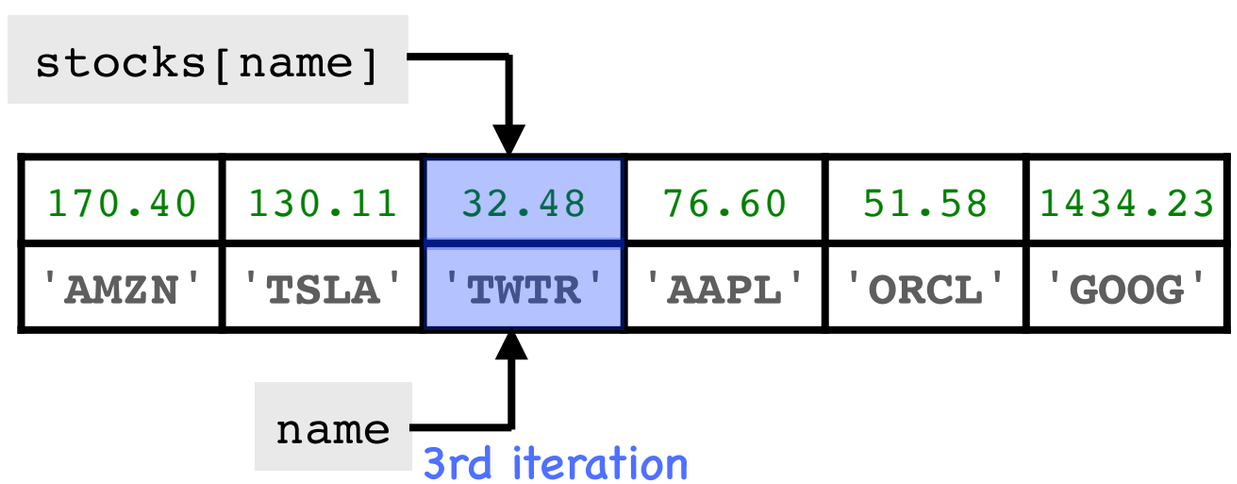


Dictionaries

- Loops and iterations with dictionaries
  - Iterating over keys

```
for name in stocks:
    print('{}: {}'.format(name, stocks[name]))
```

```
AMZN: 170.40
TSLA: 130.11
TWTR: 32.48
AAPL: 76.6
ORCL: 51.58
GOOG: 1434.23
```

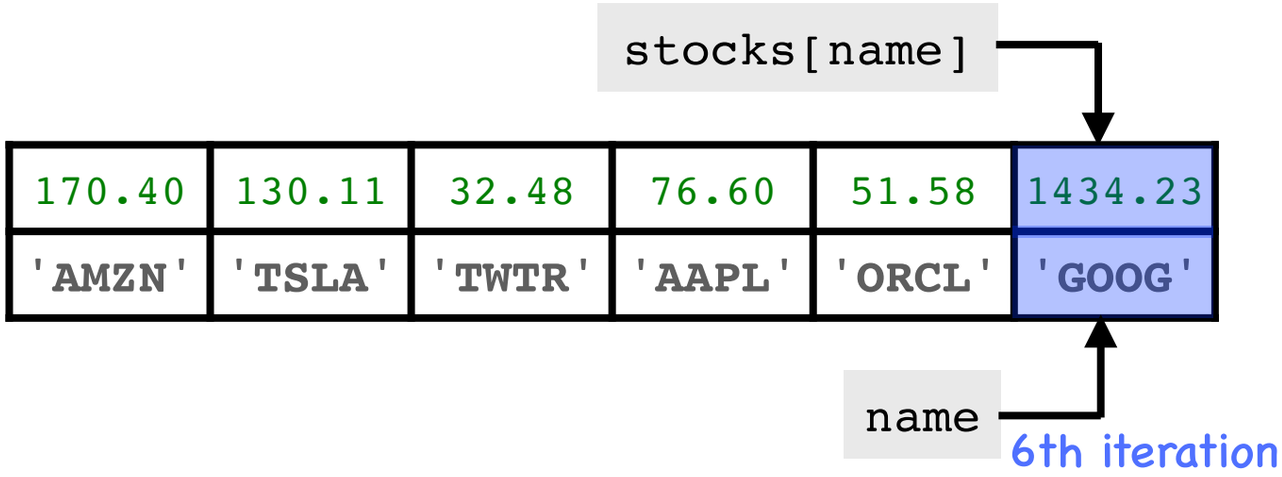


Dictionaries

- Loops and iterations with dictionaries
  - Iterating over keys

```
for name in stocks:
    print('{}: {}'.format(name, stocks[name]))
```

```
AMZN: 170.40
TSLA: 130.11
TWTR: 32.48
AAPL: 76.6
ORCL: 51.58
GOOG: 1434.23
```



Dictionaries

- Loops and iterations with dictionaries
  - Iterating over values

```
for price in stocks.values():
    print(price)
```

```
170.40
130.11
32.48
76.6
51.58
1434.23
```

`values()` method

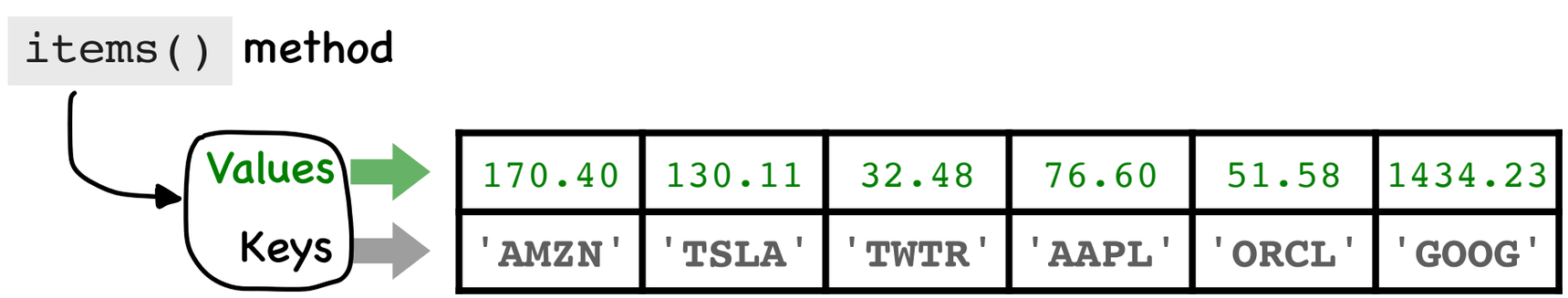
170.40	130.11	32.48	76.60	51.58	1434.23
'AMZN'	'TSLA'	'TWTR'	'AAPL'	'ORCL'	'GOOG'

Dictionaries

- Loops and iterations with dictionaries
  - Iterating over key-value items

```
for item in stocks.items():
    print(item)
```

```
('AMZN', 170.40)
('TSLA', 130.11)
('TWTR', 32.48)
('AAPL', 76.6)
('ORCL', 51.58)
('GOOG', 1434.23)
```

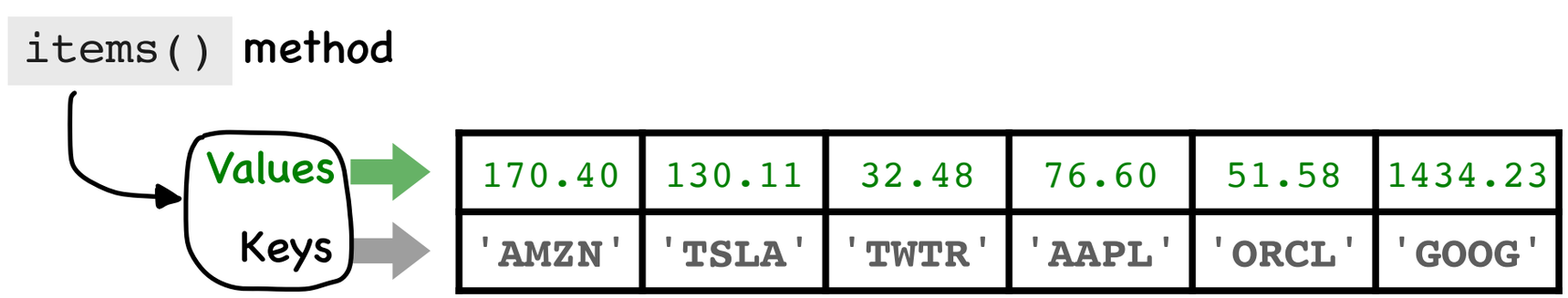


Dictionaries

- Loops and iterations with dictionaries
  - Iterating over key-value items

```
for name, price in stocks.items():
    print('{}: {}'.format(name, price))
```

```
AMZN: 170.40
TSLA: 130.11
TWTR: 32.48
AAPL: 76.6
ORCL: 51.58
GOOG: 1434.23
```

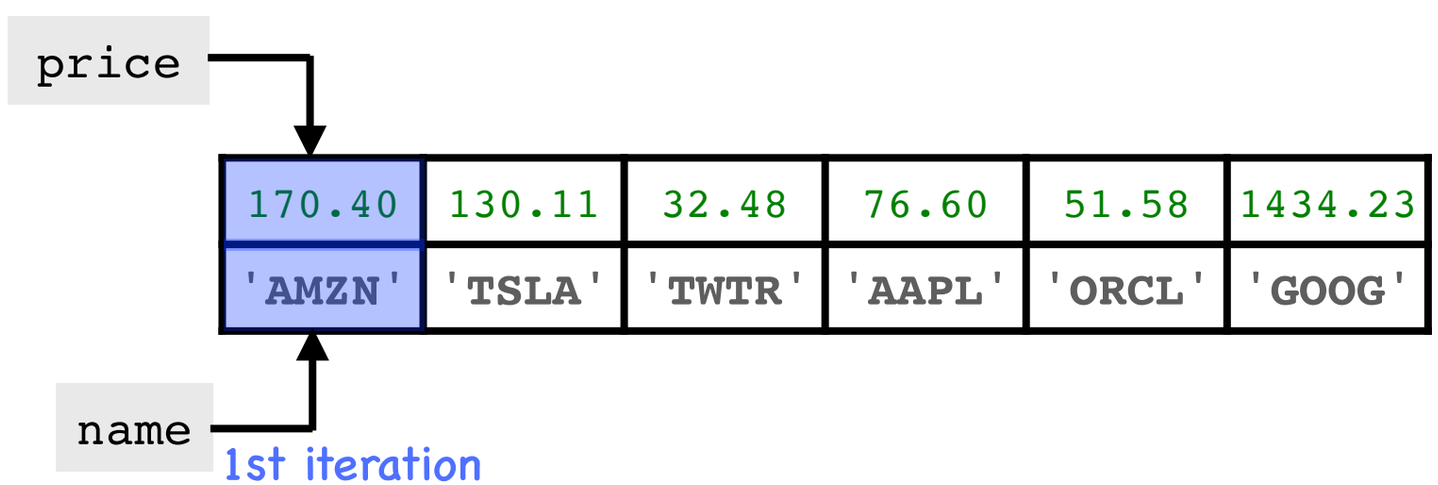


Dictionaries

- Loops and iterations with dictionaries
  - Iterating over key-value items

```
for name, price in stocks.items():
    print('{}: {}'.format(name, price))
```

```
AMZN: 170.40
TSLA: 130.11
TWTR: 32.48
AAPL: 76.6
ORCL: 51.58
GOOG: 1434.23
```

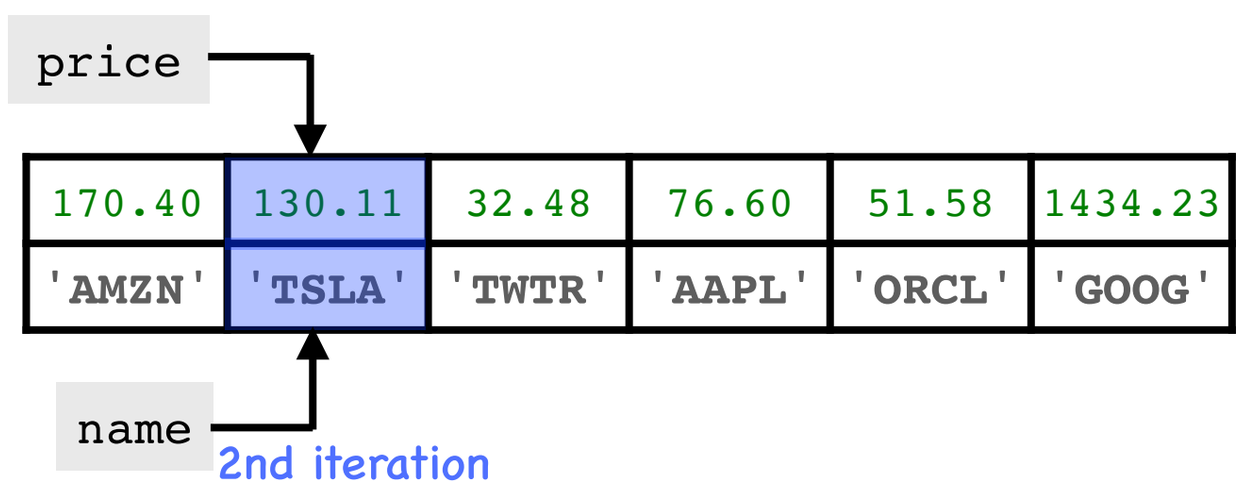


Dictionaries

- Loops and iterations with dictionaries
  - Iterating over key-value items

```
for name, price in stocks.items():
    print('{}: {}'.format(name, price))
```

```
AMZN: 170.40
TSLA: 130.11
TWTR: 32.48
AAPL: 76.6
ORCL: 51.58
GOOG: 1434.23
```

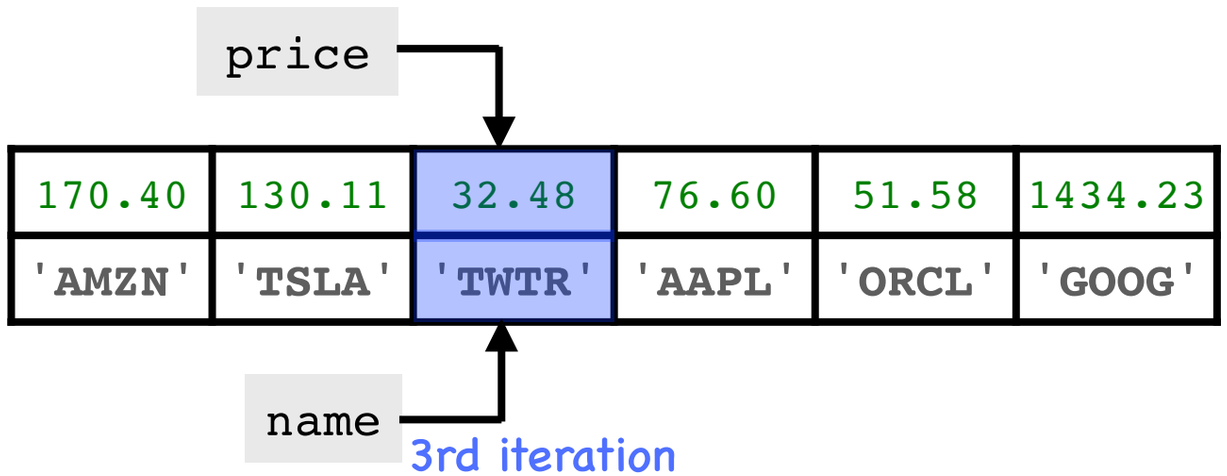


Dictionaries

- Loops and iterations with dictionaries
  - Iterating over key-value items

```
for name, price in stocks.items():
    print('{}: {}'.format(name, price))
```

```
AMZN: 170.40
TSLA: 130.11
TWTR: 32.48
AAPL: 76.6
ORCL: 51.58
GOOG: 1434.23
```

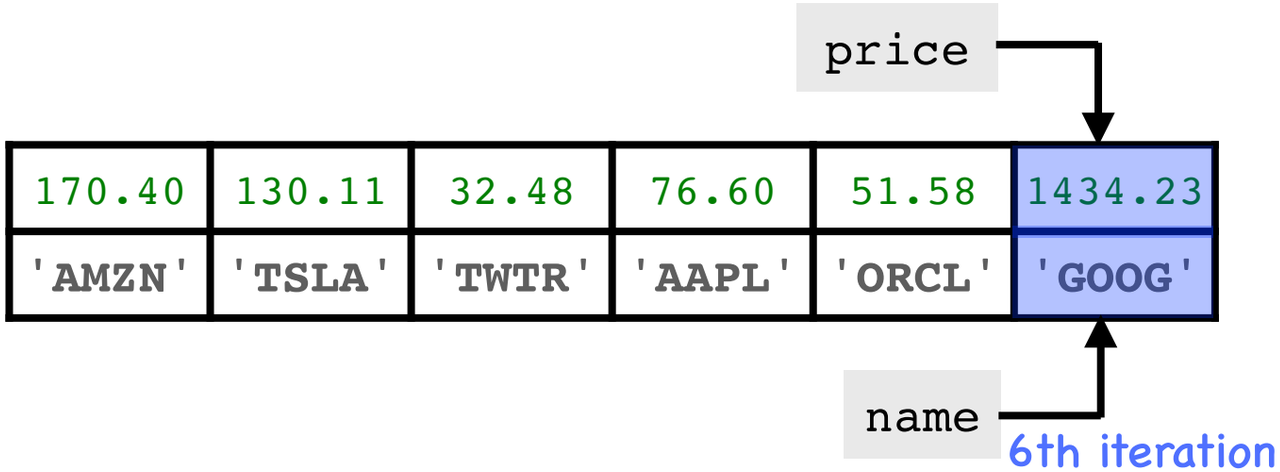


Dictionaries

- Loops and iterations with dictionaries
  - Iterating over key-value items

```
for name, price in stocks.items():
    print('{}: {}'.format(name, price))
```

```
AMZN: 170.40
TSLA: 130.11
TWTR: 32.48
AAPL: 76.6
ORCL: 51.58
GOOG: 1434.23
```



Dictionaries

- Loops and iterations with dictionaries

Question 1: The variable **`works`** is a dictionary containing a few of Wolfgang Amadeus Mozart's works, where the keys are the Köchel catalogue indexes of music works composed by Mozart, and values are the titles of these works. Write a program to group these works into two dictionaries: **`concertos`** that contains all concertos, and **`symphonies`** that contains all symphonies.

```
works = {'K. 162': 'Symphony No. 22 in C major',
         'K. 216': 'Violin Concerto No. 3',
         'K. 218': 'Violin Concerto No. 4',
         'K. 219': 'Violin Concerto No. 5',
         'K. 550': 'Symphony No. 40 in G minor',
         'K. 551': 'Symphony No. 41 in C major, "Jupiter"'}
```

Dictionaries

- Loops and iterations with dictionaries

Question 1: The variable **`works`** is a dictionary containing a few of Wolfgang Amadeus Mozart's works, where the keys are the Köchel catalogue indexes of music works composed by Mozart, and values are the titles of these works. Write a program to group these works into two dictionaries: **`concertos`** that contains all concertos, and **`symphonies`** that contains all symphonies.

```
concertos = {}
symphonies = {}

for ... ...:
    ...
    ...
    ...
    ...
```

Dictionaries

- Loops and iterations with dictionaries

Question 1: The variable **`works`** is a dictionary containing a few of Wolfgang Amadeus Mozart's works, where the keys are the Köchel catalogue indexes of music works composed by Mozart, and values are the titles of these works. Write a program to group these works into two dictionaries: **`concertos`** that contains all concertos, and **`symphonies`** that contains all symphonies.

```
concertos = {}
symphonies = {}

for key, value in works.items():
    ...
    ...
    ...
    ...
```

Dictionaries

- Loops and iterations with dictionaries

Question 1: The variable **`works`** is a dictionary containing a few of Wolfgang Amadeus Mozart's works, where the keys are the Köchel catalogue indexes of music works composed by Mozart, and values are the titles of these works. Write a program to group these works into two dictionaries: **`concertos`** that contains all concertos, and **`symphonies`** that contains all symphonies.

```
concertos = {}
symphonies = {}

for key, value in works.items():
    if ... ...:
        concertos[key] = value
    else:
        symphonies[key] = value
```

Dictionaries

- Loops and iterations with dictionaries

Question 1: The variable **`works`** is a dictionary containing a few of Wolfgang Amadeus Mozart's works, where the keys are the Köchel catalogue indexes of music works composed by Mozart, and values are the titles of these works. Write a program to group these works into two dictionaries: **`concertos`** that contains all concertos, and **`symphonies`** that contains all symphonies.

```
concertos = {}
symphonies = {}

for key, value in works.items():
    if 'Concerto' in value:
        concertos[key] = value
    else:
        symphonies[key] = value
```

Summary

- Built-in data structures

	String	List	Tuple	Dictionary
mutable	No	Yes	No	Yes
indexing and slicing	integers	integers	integers	key names
operators `+` and `*`	Yes	Yes	Yes	No
iterable	Yes	Yes	Yes	Yes
methods	Yes	Yes	No	Yes

Summary

- Parentheses and brackets

	`()`	`[]`	`{}`
Usage	1. Enclose input arguments of function and method 2. Create tuples	1. Create lists 2. Indexing and slicing	1. Dictionary and Set 2. Used in f-strings or `format()` method

Examples	`print('Hello')`
	`string.upper()`
	Empty tuple `()`
		`[1, 2, 3]`
		`string[3:]`
		`dictionary['key']`
			`{'key': 'value'}`

Remarks	1. Cannot be omitted even when there is no input argument 2. Can be omitted when creating tuples	-	Set is not covered in this course