Iterables and Iteration

Austin Bingham
@austin_bingham
austin@sixty-north.com

Presenter

Robert Smallshire
@robsmallshire
rob@sixty-north.com

pluralsight
hardcore dev and IT training

comprehensions

short-hand syntax for creating collections and iterable objects

Comprehensions can use multiple input sequences and multiple if-clauses

Benefits of comprehensions

- Container populated “atomically”
- Allows Python to optimize creation
- More readable

Comprehensions can be nested inside other comprehensions

All comprehensions nest in the same way

iteration and iterables

+

building-block functions

Ideas developed in **functional programming**

Functional-style Python

map()

apply a function to every element in a sequence,
producing a new sequence

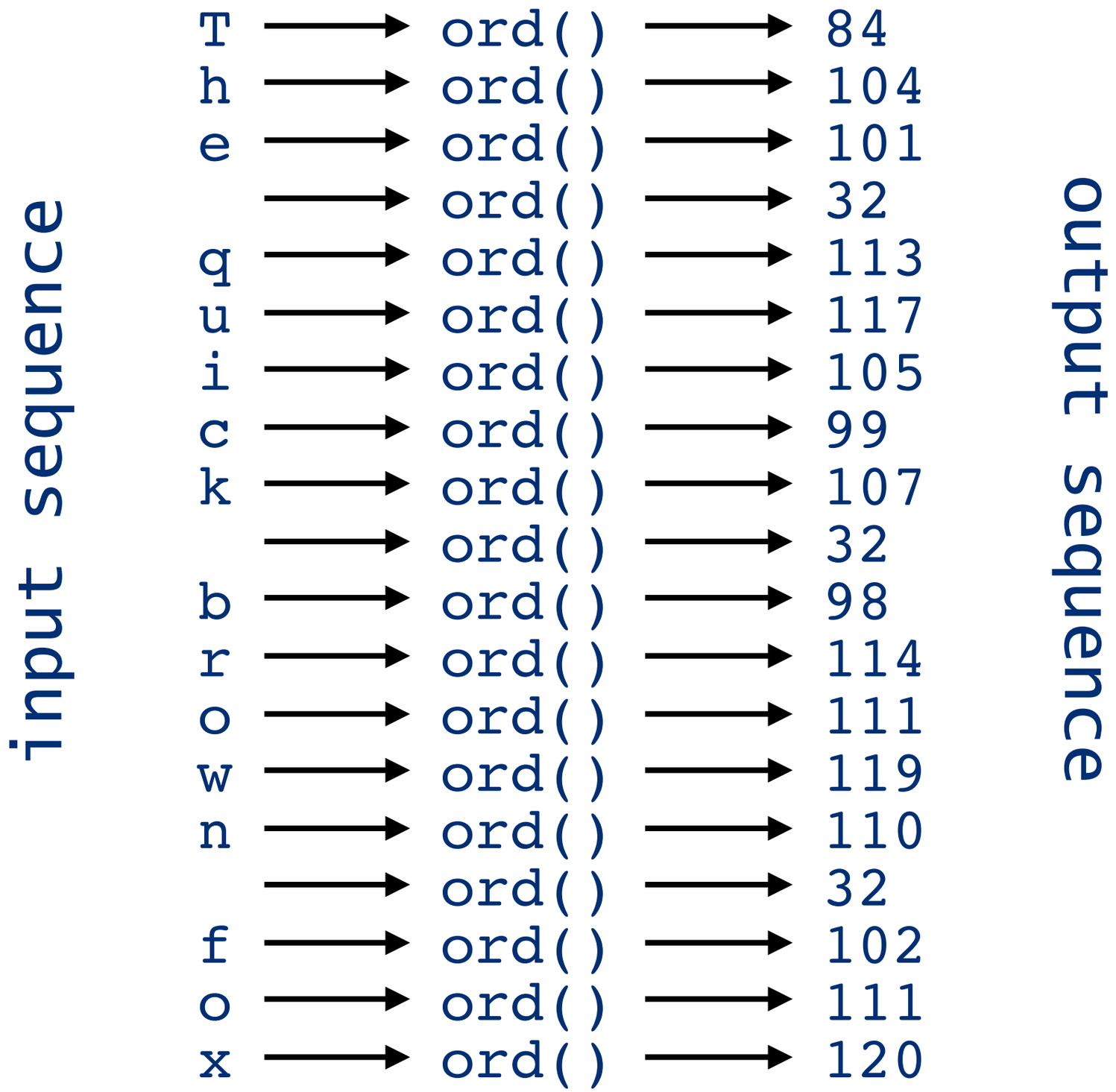
input sequence

```
map(ord, ‘The quick brown fox’)
```

function

python™

map(ord, ‘The quick brown fox’)



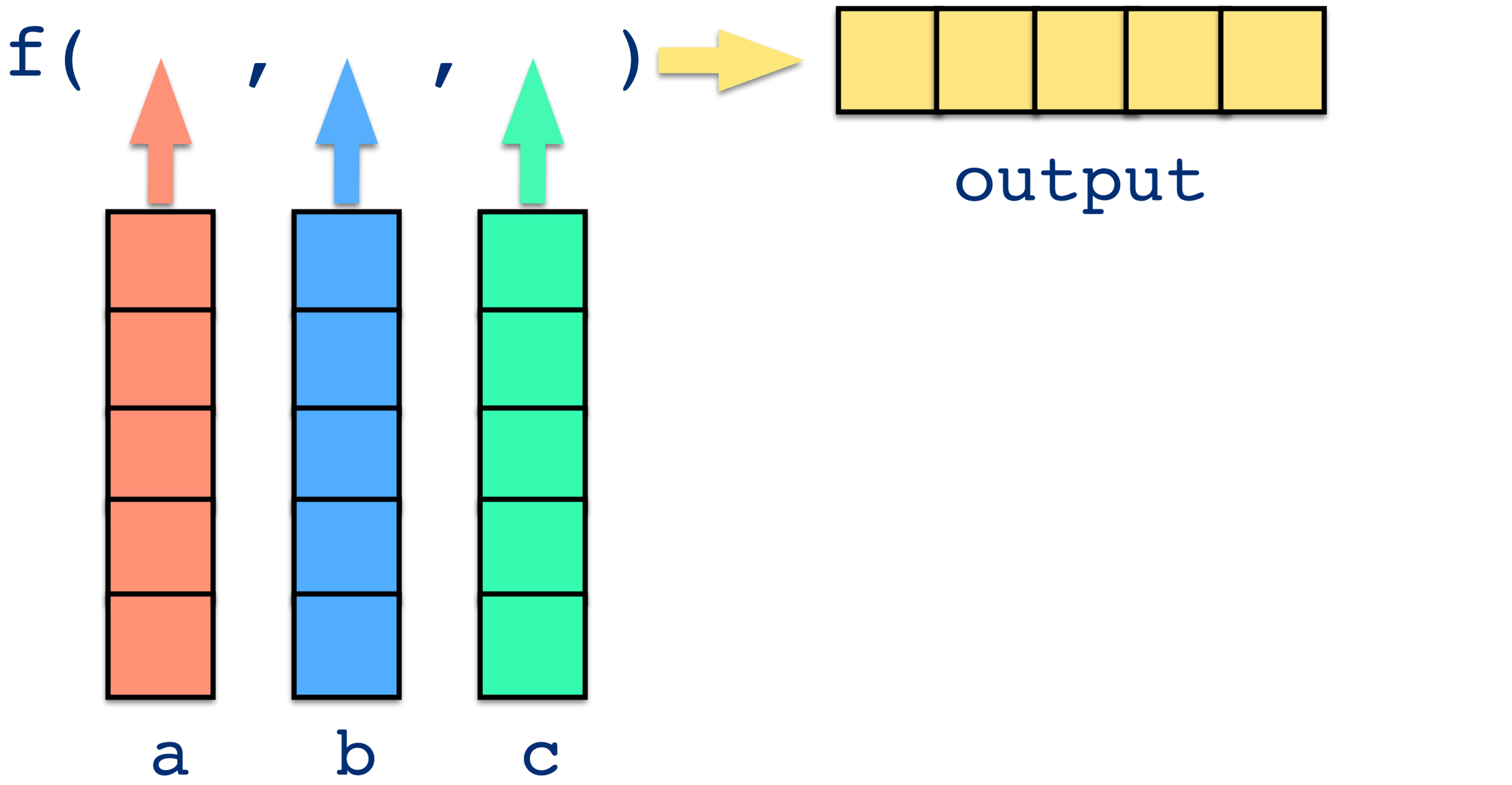
`map()` is **lazy** – it only produces values as they're **needed**

`map()` can accept any number of input sequences

The number of input sequences must match the number of function arguments

python™

map(f, a, b, c)

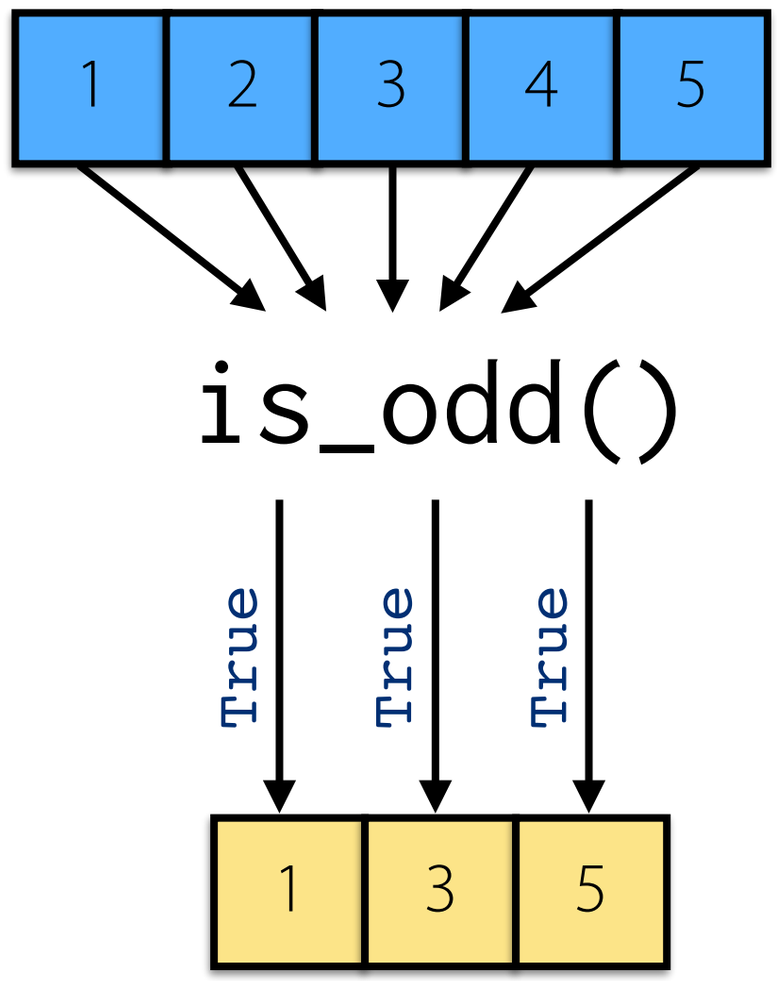


`filter()`

apply a function to each element in a sequence, constructing a new sequence with the elements for which the function returns `True`

python™

```
filter(is_odd, [1, 2, 3, 4, 5])
```



Passing `None` as the first argument to `filter()` will remove elements which evaluate to `False`.

In Python 2 `map()` and `filter()` are eagerly evaluated and return `list` objects.

Functional-programming
fold

functools.reduce()

repeatedly apply a function to the elements of a sequence,
reducing them to a single value

LINQ
aggregate()

C++ STL
std::accumulate()

Optional **initial value** is conceptually just **added** to the start of the input sequence.

The standard library `operator` module contains **function** equivalents of the **infix operators**

`a + b`

is equivalent to

`operator.add(a, b)`

`map()` and `reduce()`

=

map-reduce

Python iteration

`iter()`

create an iterator

`next()`

get next element in sequence

`StopIteration`

signal the end of the sequence

iterable

an object which implements the `__iter__()` method

iterator

an object which implements the ***iterable protocol***
and which implements the `__next__()` method

The alternative iterable protocol works with any object that supports consecutive integer indexing via `__getitem__()`

Extended `iter()`

Iteration stops when `callable` produces this value

```
iter(callable, sentinel)
```

Callable object that takes zero arguments

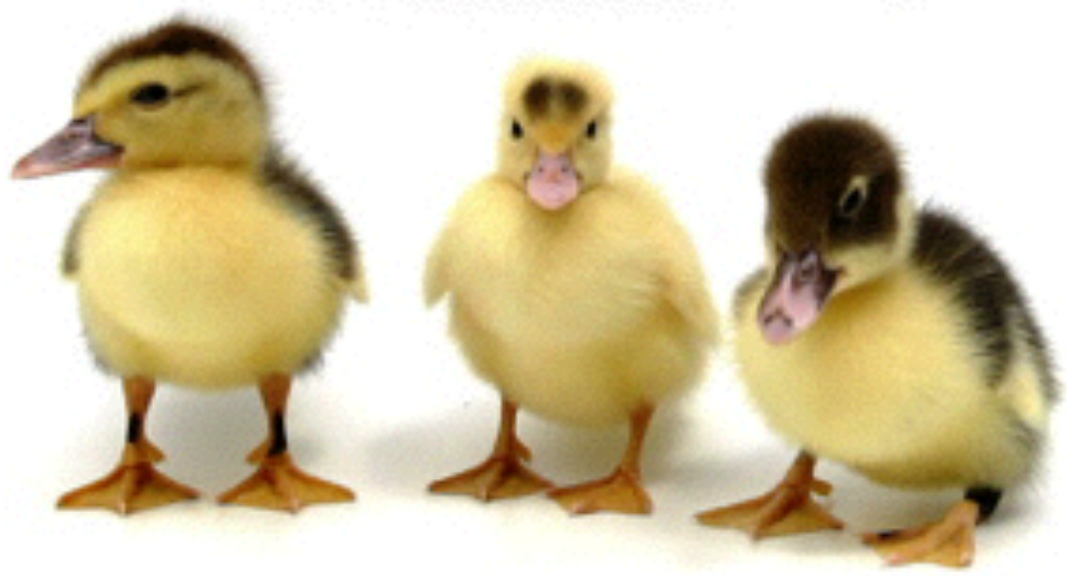
Extended `iter()` is often used for creating infinite sequences from existing functions

The return value of extended `iter()` is both an iterator and iterable

Tales of real-world Python
– better in practice than in theory –
just like Duck Typing.

Duck Tails

Real World Iterables - Sensor Data



Duck Tails

Summary: Iterables and Iteration

- Comprehensions can process more than one input sequence
- Multiple input sequences in comprehensions work like nested for-loops
- Comprehensions can also have multiple if-clauses interspersed with the for-clauses
- Later clauses in a comprehension can reference variables bound in earlier clauses
- Comprehension can also appear in the result expression of a comprehension, resulting in nested sequences
- Python provides a number of functional-style tools for working with iterators
- `map()` calls a function for each element in its input sequences
- `map()` returns an iterable object, not a fully-evaluated collection
- `map()` results are lazily evaluated, meaning that you must access them to force their calculation
- `map()` results are typically evaluated through the use of iteration constructs such as for-loops

Summary: Iterables and Iteration

- You must provide as many input sequences to `map()` as the callable argument has parameters
- `map()` takes one element from each input sequence for each output element it produces
- `map()` stops producing output when its shortest input sequence is exhausted
- `map()` can be used to implement the same behavior as comprehensions in some cases
- `filter()` selects values from an input sequence which match a specified criteria
- `filter()` passes each element in its input sequence to the function argument
- `filter()` returns an iterable over the input elements for which the function argument is *truthy*
- Like `map()`, `filter()` produces its output lazily
- If you pass `None` as the first argument to `filter()`, it yields the input values which evaluate to `True` in a boolean context

Summary: Iterables and Iteration

- `reduce()` cumulatively applies a function to the elements of an input sequence
- `reduce()` calls the input function with two arguments: the accumulated result so far, and the next element in the sequence
- `reduce()` is a generalization of summation
- `reduce()` returns the accumulated result after all of the input has been processed
- If you pass an empty sequence to `reduce()` it will raise a `TypeError`
- `reduce()` accepts an optional initial value argument
  - This initial value is conceptually added to the front of the input sequence
- The initial value is returned if the input sequence is empty
- The `map()` and `reduce()` functions in Python are related to the ideas in the map-reduce algorithm

Summary: Iterables and Iteration

- Python's `next()` function calls `__next__()` on its argument
- Iterators in Python must support the `__next__()` method
- `__next__()` should return the next item in the sequence, or raise `StopIteration` if it is exhausted
- Python's `iter()` function calls `__iter__()` on its argument
- Iterable objects in Python must support the `__iter__()` method
- `__iter__()` should return an iterator for the iterable object
- Objects with a `__getitem__()` method that accepts consecutive integer indices starting at zero are also iterables
- Iterables implemented via `__getitem__()` must raise `IndexError` when they are exhausted
- The extended form of `iter()` accepts a zero-argument callable and a sentinel value
- Extended `iter()` repeatedly calls the callable argument until it returns the sentinel value

Summary: Iterables and Iteration

- The values produced by extended `iter()` are those returned from the callable
- One use case for extended `iter()` is to iterate using simple functions
- Protocol conforming iterators must also be iterable