Computer Code for Beginners
Week 5

Matt Luckcuck

19th October 2017

Last Time

Previously...

- Functions
- Sequences
  - List
  - String
  - Tuple

Outline

Outline

- Dictionaries

Sequence Recap

Sequences

List Recap

- A sequence is an ordered set of data
- Zero-Indexed
- List
  - `colours = ["Red", "Blue", "Green"]`
  - Mutable
- String
  - `s = "Purple"`
  - Immutable
- Tuple
  - `t = (170,52)`
  - Immutable

Sequences

String

- Another sequence type
  - Each character is indexed by a number
- `s = "Purple"`
- Allows some sequence operations:
  - Indexing – `s[0]`
  - Slicing – `s[0:2]`
- But Strings are *immutable*

Sequences

String

- Another sequence type
  - Each character is indexed by a number
- `s = "Purple"`
- Allows some sequence operations:
  - Indexing – `s[0]`
    - `"P"`
  - Slicing – `s[0:2]`
- But Strings are *immutable*

Sequences

String

- Another sequence type
  - Each character is indexed by a number
- `s = "Purple"`
- Allows some sequence operations:
  - Indexing – `s[0]`
    - `"P"`
  - Slicing – `s[0:2]`
    - `"Pu"`
- But Strings are *immutable*

Sequences

Tuple

- Another ordered set of data
- Cannot be changed (immutable)
  - We say this is *immutable*
- Zero-indexed like a list or string
- `t = (170,52)` Simple Tuple (Pair)

Sequences

Tuple

- Another ordered set of data
- Cannot be changed (immutable)
  - We say this is *immutable*
- Zero-indexed like a list or string
- `t = (170,52)` Simple Tuple (Pair)
- `t[0]` is?

Sequences

Tuple

- Another ordered set of data
- Cannot be changed (immutable)
  - We say this is *immutable*
- Zero-indexed like a list or string
- `t = (170,52)` Simple Tuple (Pair)
- `t[0]` is?
  - 170

Sequences

Tuple

- Another ordered set of data
- Cannot be changed (immutable)
  - We say this is *immutable*
- Zero-indexed like a list or string
- `t = (170,52)` Simple Tuple (Pair)
- `t[0]` is?
  - 170
- `t[1]` is?

Sequences

Tuple

- Another ordered set of data
- Cannot be changed (immutable)
  - We say this is *immutable*
- Zero-indexed like a list or string
- `t = (170,52)` Simple Tuple (Pair)
- `t[0]` is?
  - 170
- `t[1]` is?
  - 52

Sequences

Tuple

- Another ordered set of data
- Cannot be changed (immutable)
  - We say this is *immutable*
- Zero-indexed like a list or string
- `t = (170,52)` Simple Tuple (Pair)
- `t[0]` is?
  - 170
- `t[1]` is?
  - 52
- What about `t[0] = 7`?

Sequences

Tuple

- Another ordered set of data
- Cannot be changed (immutable)
  - We say this is *immutable*
- Zero-indexed like a list or string
- `t = (170,52)` Simple Tuple (Pair)
- `t[0]` is?
  - 170
- `t[1]` is?
  - 52
- What about `t[0] = 7`?
  - `TypeError: "tuple"object does not support item assignment`

Sequences

Tuples

- Useful for passing around related data
  - Height and Weight (Pair)
  - X, Y, and Z coordinates (Triple)
  - Car tyre pressures (4 Tuple)
- Again, using them is a design decision
- Multiple returns...
  - Reminder: `return x, y`
  - This is returning a tuple
  - But if there are a lot of values to return...
  - `return (x,y,z,a,b,c,d,e)` or
    `return (x,y,z), (a,b,c), (d,e)`
  - Design decision

Dictionaries

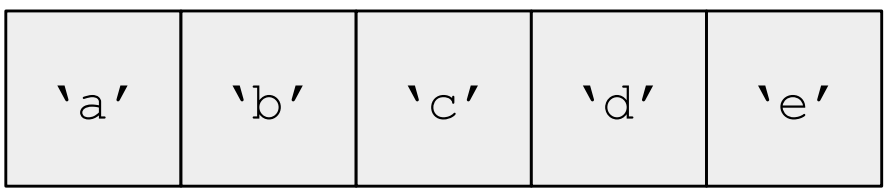
Dictionaries

Dictionaries $\neq$ Sequences

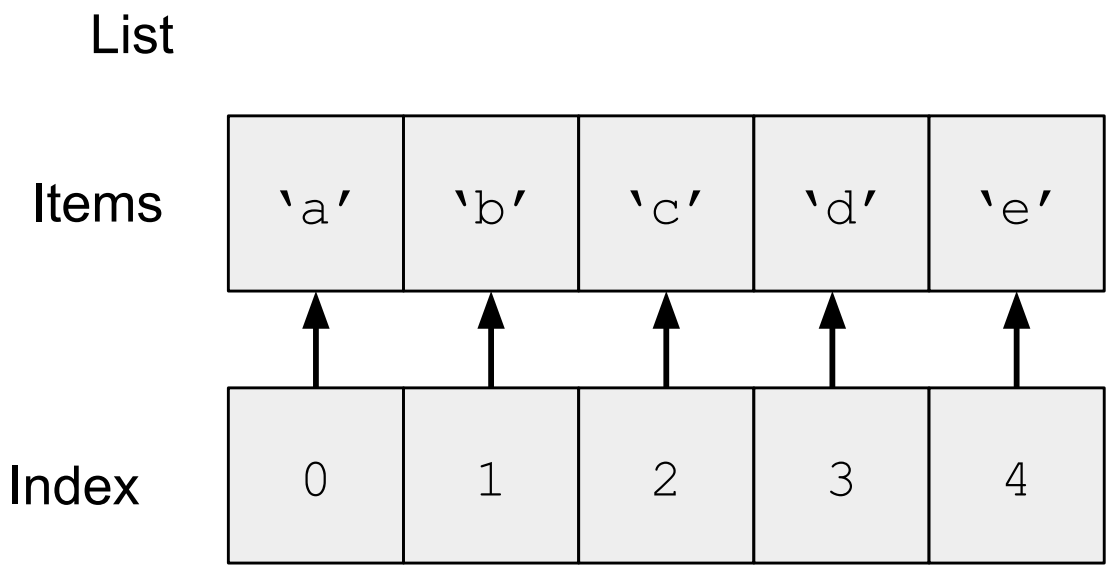
- Unordered set of data
- `d = {"Octopus":42,"cat": 5}`
  - Sometimes called *Maps* or *Associative Arrays*
- Maps `keys` to `values`
  - Think of two lists, or
  - Think of a set of pairs (Key, Value)
- Each item is a key and a value

Dictionaries

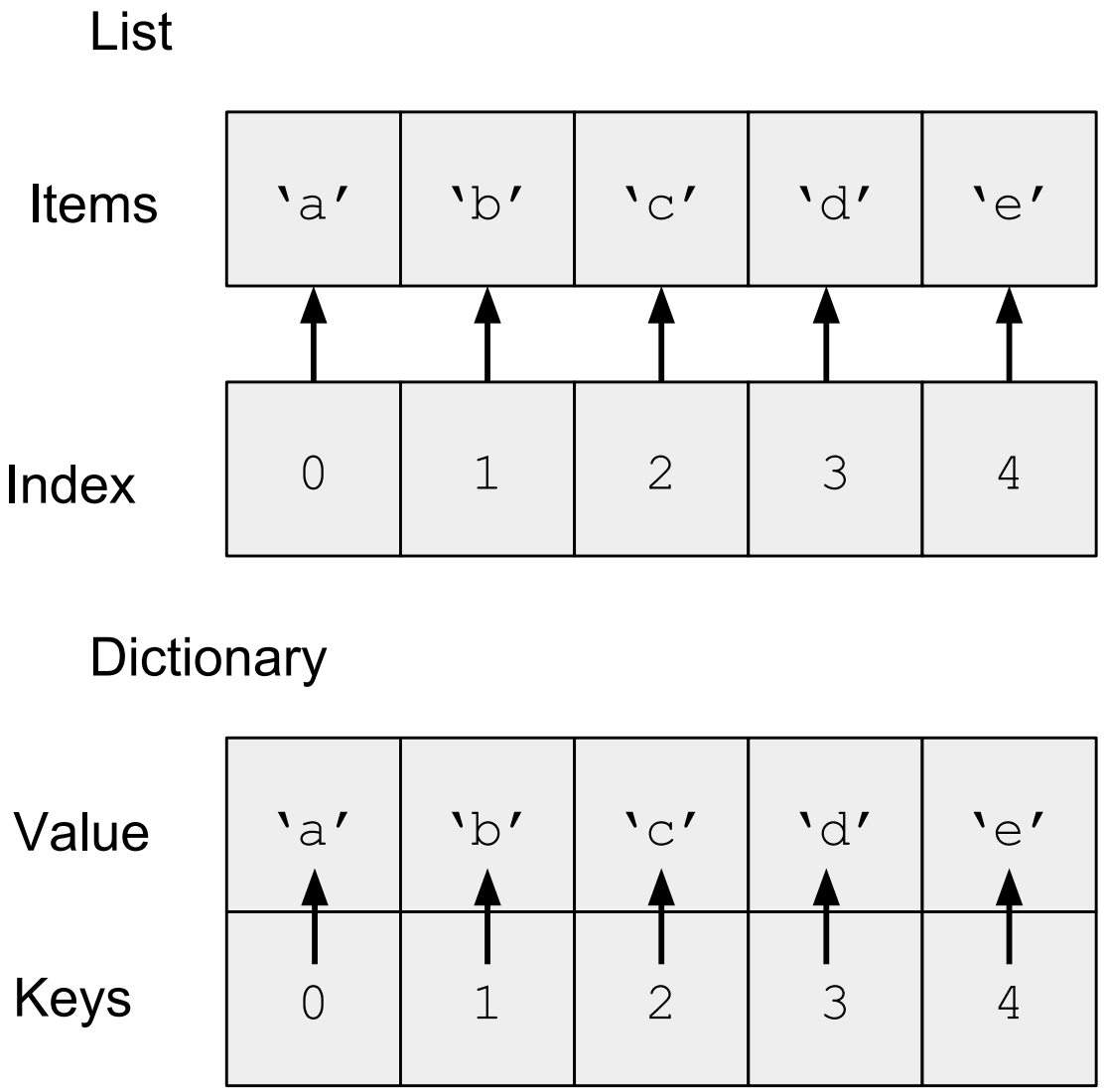
List



Dictionaries



Dictionaries



Dictionaries

Dictionary Keys

- Keys can be any immutable type:
  - String
  - Number
  - Tuples (only containing Strings, Numbers, or Tuples)
- Keys must be unique in one Dictionary

Dictionaries

Useful Dictionary Operations

- `d = dict()` or `d = {}` makes a new empty dictionary
- `d = {"a":7, "b":12}` makes a dictionary with entries
  - or `d = dict([("a", 7), ("b", 12)])`
- `d["b"]` gives us 7
- `d["c"] = 5` adding an entry
- `d.keys()` Lists the keys
- `d.values()` Lists the values
- `d.items()` lists tuples of key, value pairs
  - Order unreliable

Summary

Summary

Summary

- Dictionaries
  - Map keys to values
- Exceptions
  - Catching errors
- File Handling
- JSON Format

Exercises

- Letters in a String
  - Using a dictionary to keep track of how many of each different letter there are in a string
- Morse Code
  - Dictionary of letters mapping to their Morse code