Lecture 12 - Data Structures (Part 2)

DSE 511

Drew Schmidt
2022-10-04

Announcements

- Nothing unresolved from last time
- Homework is live!
  - Due date moved to Wed Oct 5 (midnight)
  - New homework *next* week!
- No class Thursday Oct 6
- Questions?

What's The Goal

- Better code organization
- Engineering over scripting
- Encapsulation over globals

Content

- Lists
- Classes
- Dictionaries

Lists

Lists

- An ordered collection of *objects*
- Can store anything
  - Arrays
  - Dataframes
  - Other lists
  - Custom class objects
- Can be implemented in different ways

Lists vs Arrays

Lists

Ordered collection of a objects.

Arrays

Ordered collection of a values from a single fundamental type.

Lists in R and Python

R

```r
x = list(1, "2", list(a=1, b=2))
x
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] "2"
##
## [[3]]
## [[3]]$a
## [1] 1
##
## [[3]]$b
## [1] 2
```

Python

```python
x = [1, '2', {'a':1, 'b':2}]
```

```
[1, '2', {'a': 1, 'b': 2}]
```

Lists

- Extremely important in R
  - Easy way to handle multi-value returns
  - S3 class decorator/generics
  - ...
- A place to put things
- Helpful for managing lots of things programmatically

Classes

Classes and Objects

- Object Oriented Programming (OOP)
  - Useful organizational strategy
  - Not the only paradigm
- A *class* is a custom data type
  - Describes the data
  - Defines the methods
- It describes an *object*, or instantiation of the class
- The methods operate on the objects
- Methods may use side-effects

Pseudocode

```
x = create_class_object()
x.compute()
y = x.get_result()
```

OOP Systems

R

- S3
- S4
- R6 (CRAN package)

Python

- The `class` system

Why Classes?

- Encapsulation
- No ambiguity over function arguments
- Solves the "state problem"

Encapsulation

- Only an object has access to class methods
- Same method name across different classes do different things
- Don't have to worry about unifying different objects under one API
- Useful if you're bad at naming things!

Function Arguments

Without OO

```
x = construct_object()

f = function(x) {
  # ...
}
x = f(x)
```

With OO

```
x = NewObject()
x$f()
```

State

- Try not to use global variables
- But what if you have a lot of "moving pieces"

```
my_fun(x, y, z, w, a, b, c, d, e, ...)
```

- Maybe you can re-organize as a class
  - Stuff the data into the object
  - Operate on options internally

```
x$my_fun()
```

Customer Class

```
Customer = R6::R6Class(
  public = list(
    initialize = function() {
      self$id = private$create_id()
      self$orders = 0
      self$spend = 0
    },
    increment_orders = function() {
      self$orders = self$orders + 1
    },
    increment_spend = function(spend) {
      self$spend = self$spend + spend
    },
    id = NULL, orders = NULL, spend = NULL
  ),
  private = list(
    create_id = function() {
      httr::GET(url = "https://www.uuidgen
        httr::content()
    }
```

```
customer = Customer$new()
customer$id
```

```
## [1] "5d432dee-564f-44aa-a72d-881d3b8f27fd"
```

```
customer$increment_orders()
customer$orders
```

```
## [1] 1
```

```
new_spend = 12.34
customer$increment_spend(new_spend)
customer$spend
```

```
## [1] 12.34
```

Dictionaries

Dictionaries

- Key/value storage
- Like an in-memory json
- Indexed by keys (not necessarily numbers)
- Major operations
  - Insertion
  - Deletion
  - Retrieval
- Often implemented as a hash table

Hash Table vs Lists

What's the value for the key `"434c8fb1-c329-4234-891d-60bf0a14791e"` ?

List

- Check the first element
- Check the second element
- ...

Hash Table

- Run key through "hash function"
- Get value at index output by hash function

Hash Table vs Lists

What's the value for the key `"434c8fb1-c329-4234-891d-60bf0a14791e"` ?

List

Runtime scales with the number of elements in the list

Hash Table

Runtime is fixed regardless of the size of the hash table

Dictionaries in Python

```
d = {}
d['key1'] = 'val1'
d['key2'] = 'val2'
d
```

```
{'key1': 'val1', 'key2': 'val2'}
```

```
d['key1']
d['key1'] = 'newval'
d
```

```
{'key1': 'newval', 'key2': 'val2'}
```

Building a Dictionary in R

- R has no "dictionary" built in
- It does have hash tables (environments)
- We can implement a dictionary using a hash table...

Dictionary

```
Dictionary = R6::R6Class(
  public = list(
    initialize = function() {
      private$env = new.env()
      invisible(self)
    },
    length = function() {
      base::length(private$env)
    },
    print = function() {
      cat("## A dictionary with", self$le
    }
  ),
  private = list(
    env = NULL
  )
)
```

```
d = Dictionary$new()
d
```

```
## A dictionary with 0 elements
```

Dictionary

```
  # ...
  insert = function(key, value) {
    private$env[[key]] = value
    invisible(self)
  },
  get = function(key) {
    private$env[[key]]
  },
  # ...
```

```
d = Dictionary$new()
d
```

```
## A dictionary with 0 elements
```

```
d$insert("x", 1)
d$get("x")
```

```
## [1] 1
```

```
d
```

```
## A dictionary with 1 elements
```

Dictionary

```
  # ..
  keys = function() {
    ls(private$env)
  },
  values = function() {
    as.list(private$env)
  }
  # ..
```

```
d = Dictionary$new()
d$insert("x", 1)
d$values()
```

```
## $x
## [1] 1
```

Dictionary

```
  # ...
  update = function(dict) {
    keys_new = dict$keys()
    for (key in keys_new)
      self$insert(key, dict$get(key))
  },
  # ...
```

```
d = Dictionary$new()
d$insert("x", 1)

new_dict = Dictionary$new()
new_dict$insert("y", 2)
d$update(new_dict)

d
```

```
## A dictionary with 2 elements
```

```
d$values()
```

```
## $x
## [1] 1
##
## $y
## [1] 2
```

Dictionary

```
  # ...
  clear = function() {
    rm(private$env)
    invisible(gc())
    private$env = new.env()
  },
  # ...
```

```
d = Dictionary$new()
d$insert("x", 1)
d
```

```
## A dictionary with 1 elements
```

```
d$clear()
d
```

```
## A dictionary with 0 elements
```

```
d$values()
```

```
## list()
```

A Dictionary of Customers

```
create_customer = function(customers) {
  customer = Customer$new()
  customers$insert(customer$id, customer)
  customers
}

customers = Dictionary$new()
for (i in 1:5) {
  customers = create_customer(customers)
}

keys = customers$keys()
keys
```

```
## [1] "434c8fb1-c329-4234-891d-60bf0a14791e"
## [2] "723188e1-c0f1-4b17-8ac8-61621405effd"
## [3] "75a7edbb-98f7-4cd4-bd71-c56c90b2a97d"
## [4] "76700f38-54e9-4194-b0d1-1ecb78812336"
## [5] "f1a3e082-f09e-4b81-8168-898fd1845b44"
```

A Dictionary of Customers

```
customer_id = keys[1]
customer = customers$get(customer_id)

customer$increment_orders()
customer$orders
```

```
## [1] 1
```

```
new_spend = 7.89
customer$increment_spend(new_spend)
customer$spend
```

```
## [1] 7.89
```

```
customers$insert(customer_id, customer)
customers$get(customer_id)$spend
```

```
## [1] 7.89
```

Wrapup

Wrapup

- Classes
  - Can be a helpful way of organizing code
  - Not a "suicide pact"
- Lists and dictionaries are useful data structures

Questions?