Lab 5: Data Abstraction, Sequences

Gabe Classon's CS 61A lab

9:30–11:00 a.m. Wednesday, Feb. 22, 2023

Question of the day

What's the last worst meal you had?

Announcements

- https://tinyurl.com/UAW-EECS-survey
- Academic interns mini-lecturing

Lab 05

- Q1: Flatten: fundamental list manipulation
- Q2–Q4: Map, Filter, and Reduce: list compeehensions, common techniques for dealing with lists
- Q5: Distance: ADTs, selectors
- Q6: Closer City: extending an interface

Sophia Liu

Data Abstraction

What is abstraction?

- You don’t need to know exactly how something works to be able to use it!

What is abstraction?

- You don’t need to know exactly how something works to be able to use it!
    - Driving a car: Knowing how the mechanics behind pushing the brake pedal is able to stop the car (like the pistons and the whatever IDK???)

What is abstraction?

- You don’t need to know exactly how something works to be able to use it!
    - Driving a car: ~~Knowing how the mechanics behind pushing the brake pedal is able to stop the car (like the pistons and the whatever IDK???)~~ Only need to know that the brake pedal will stop the car when you need it to

What is abstraction?

- You don’t need to know exactly how something works to be able to use it!
    - Driving a car: ~~Knowing how the mechanics behind pushing the brake pedal is able to stop the car (like the pistons and the whatever IDK???)~~ Only need to know that the brake pedal will stop the car when you need it to
- Abstraction separates the underlying implementation and the use – this is the **abstraction barrier**!

Compound Values

- Compound values combine multiple values together as objects!

Compound Values

- Compound values combine multiple values together as objects!
    - **A date**: the month, day, and year

      February 2nd, 2023 **->** `today = date('February', 2, 2023)`

Compound Values

- Compound values combine multiple values together as objects!
  - **A date**: the month, day, and year

    February 2nd, 2023 **->** `today = date('February', 2, 2023)`
  - **A car**: the make, model, and year

    1987 Buick Grand National **->** `my_car = car('Buick', 'Grand National', 1987)`

Compound Values

- Compound values combine multiple values together as objects!
  - **A date**: the month, day, and year

    February 2nd, 2023 **->** `today = date('February', 2, 2023)`
  - **A car**: the make, model, and year

    1987 Buick Grand National **->** `my_car = car('Buick', 'Grand National', 1987)`
  - **A geographic location**: the latitude and longitude

    Soda Hall (37.8756° N, 122.2588° W) **->** `soda_hall = location(37.8756, 122.2588)`

Compound Values

- Compound values combine multiple values together as objects!
  - **A date**: the month, day, and year

    February 2nd, 2023 **->** `today = date('February', 2, 2023)`
  - **A car**: the make, model, and year

    1987 Buick Grand National **->** `my_car = car('Buick', 'Grand National', 1987)`
  - **A geographic location**: the latitude and longitude

    Soda Hall (37.8756° N, 122.2588° W) **->** `soda_hall = location(37.8756, 122.2588)`
- Question: Can you think of a compound value/object that we've worked with a lot recently?

Compound Values

- Compound values combine multiple values together as objects!
  - **A date**: the month, day, and year

    February 2nd, 2023 **->** `today = date('February', 2, 2023)`
  - **A car**: the make, model, and year

    1987 Buick Grand National **->** `my_car = car('Buick', 'Grand National', 1987)`
  - **A geographic location**: the latitude and longitude

    Soda Hall (37.8756° N, 122.2588° W) **->** `soda_hall = location(37.8756, 122.2588)`
- Question: Can you think of a compound value/object that we've worked with a lot recently? **LISTS**

NOT Violating the Abstraction Barrier

- Like the driver of car, the end user of a data abstraction does not need to know how the code is implemented under the hood, only **what it does and how to use it**

NOT Violating the Abstraction Barrier

- Like the driver of car, the end user of a data abstraction does not need to know how the code is implemented under the hood, only **what it does and how to use it**
- To avoid violating the abstraction barrier, we use

NOT Violating the Abstraction Barrier

- Like the driver of car, the end user of a data abstraction does not need to know how the code is implemented under the hood, only **what it does and how to use it**
- To avoid violating the abstraction barrier, we use
  - **constructors**

    `my_car = car('Buick', 'Grand National', 1987)` and
  - **selectors** `get_year(my_car)`

NOT Violating the Abstraction Barrier

- Like the driver of car, the end user of a data abstraction does not need to know how the code is implemented under the hood, only **what it does and how to use it**
- To avoid violating the abstraction barrier, we use
  - **constructors**

    `my_car = car('Buick', 'Grand National', 1987)` and
  - **selectors** `get_year(my_car)`

Do not peek under the hood!

Constructors

- Constructors are functions that **create** objects

```
my_car = car(‘Buick’, ‘Grand National’, 1987)
```

Constructors

- Constructors are functions that **create** objects

```
my_car = car('Buick', 'Grand National', 1987)
```

- `car(<make>, <model>, <year>)`: `<make>`, `<model>`, and `<year>` are the parameters of the function

Constructors

- Constructors are functions that **create** objects

```
my_car = car('Buick', 'Grand National', 1987)
```

- `car(<make>, <model>, <year>)`: `<make>`, `<model>`, and `<year>` are the parameters of the function
- How exactly a constructor for an object works is beyond our concern as users of the data abstraction – its implementation lies behind the abstraction barrier! We only USE the constructor.

Constructors

- Constructors are functions that **create** objects

```
my_car = car('Buick', 'Grand National', 1987)
```

- `car(<make>, <model>, <year>)`: `<make>`, `<model>`, and `<year>` are the parameters of the function
- How exactly a constructor for an object works is beyond our concern as users of the data abstraction – its implementation lies behind the abstraction barrier! We only USE the constructor.

Do not peek under the hood!

Selectors

- Selectors are functions that **retrieve information** about an object

Selectors

- Selectors are functions that **retrieve information** about an object

```
>>> my_car = car('Buick', 'Grand National', 1987)
```

Selectors

- Selectors are functions that **retrieve information** about an object

```
>>> my_car = car('Buick', 'Grand National', 1987)
>>> get_make(my_car)
```

Selectors

- Selectors are functions that **retrieve information** about an object

```
>>> my_car = car('Buick', 'Grand National', 1987)
>>> get_make(my_car)
'Buick'
```

Selectors

- Selectors are functions that **retrieve information** about an object

```
>>> my_car = car('Buick', 'Grand National', 1987)
>>> get_make(my_car)
'Buick'
>>> get_model(my_car)
'Grand National'
>>> get_year(my_car)
1987
```

Selectors

- Selectors are functions that **retrieve information** about an object

```
>>> my_car = car('Buick', 'Grand National', 1987)
>>> get_make(my_car)
'Buick'
>>> get_model(my_car)
'Grand National'
>>> get_year(my_car)
1987
```

Like constructors, we do not care about the underlying implementation of the selectors (lies behind the abstraction barrier). We only USE the selectors.

Do not peek under the hood!

Parting thoughts...

- Constructors and selectors allow us to **avoid repetitive code** and work with objects in intuitive ways (want the make of a car object? call `get_make`)

Parting thoughts...

- Constructors and selectors allow us to **avoid repetitive code** and work with objects in intuitive ways (want the make of a car object? call `get_make`)
- For this lab, the implementation of the constructor and selectors is available for you to look at (implementation = function definitions of the constructor and selectors), and you are able to look behind the abstraction barrier.

Parting thoughts...

- Constructors and selectors allow us to **avoid repetitive code** and work with objects in intuitive ways (want the make of a car object? call `get_make`)
- For this lab, the implementation of the constructor and selectors is available for you to look at (implementation = function definitions of the constructor and selectors), and you are able to look behind the abstraction barrier.
    - But even after becoming aware of their implementation, you should not write code that *relies* on information from the implementation – this is violating the abstraction barrier.

Parting thoughts...

- Constructors and selectors allow us to **avoid repetitive code** and work with objects in intuitive ways (want the make of a car object? call `get_make`)
- For this lab, the implementation of the constructor and selectors is available for you to look at (implementation = function definitions of the constructor and selectors), and you are able to look behind the abstraction barrier.
    - But even after becoming aware of their implementation, you should not write code that *relies* on information from the implementation – this is violating the abstraction barrier.
    - Instead, **use the constructor and selectors** to perform the tasks you need to do!