Computational Structures in Data Science

Abstract Data Types (ADT)

Week 3, Summer 2024. 7/3 (Wed)

Berkeley
UNIVERSITY OF CALIFORNIA

Announcements

- Project01 ("Maps") released today!
  - Tip: start early!
  - Partner project (up to one partner), or can work solo
- Holiday: no lecture/lab/OH tomorrow (Thurs, 4th of July)

Announcements: Midterm scheduling!

- IMPORTANT: Complete the "Midterm Exam Scheduling" form on Gradescope
  - Due: Tuesday July 9th, 11:59 PM PST
  - Please, please do this ASAP! Thank you!

Lecture overview

- Mutable Functions (finish off slides we didn't get to yesterday)
- Abstract Data Types ("ADT")
  - A central idea for Project01 ("Maps")

Computational Structures in Data Science

(Leftover slides from yesterday)

Berkeley
UNIVERSITY OF CALIFORNIA

Functions with state: Bank account

- Goal: Define a function to repeatedly withdraw from a bank account that starts with $100.

```
# Desired usage
# Build our account with initial $100
balance
>>> withdraw = make_withdraw_account(100)
# withdraw $25, balance is now $75
>>> withdraw(25)
75
# withdraw another $25, balance is now $50
>>> withdraw(25)
50
# Attempt to withdraw $60, but ran out
>>> withdraw(60)
'Insufficient funds'
```

```
def make_withdraw_account(initial_balance):
    # FILL ME IN
```

Question: how to implement `make_withdraw_account()`?

Functions with state: Bank account

- Goal: Define a function to repeatedly withdraw from a bank account that starts with $100.

```
# Desired usage
# Build our account with initial $100
balance
>>> withdraw = make_withdraw_account(100)
# withdraw $25, balance is now $75
>>> withdraw(25)
75
# withdraw another $25, balance is now $50
>>> withdraw(25)
50
# Attempt to withdraw $60, but ran out
>>> withdraw(60)
'Insufficient funds'
```

```
def make_withdraw_account(initial):
    balance = [initial]
    def withdraw(amount):
        if balance[0] - amount < 0:
            return 'Insufficient funds'
        balance[0] -= amount
        return balance[0]
    return withdraw
```

A mutable value in the parent frame can maintain the local state for a function.

View in PythonTutor

Aside: Counters and nonlocal/global

- Here is another implementation. What Would Python Print?

```python
def make_counter():
    my_state = 0
    def count_up():
        my_state = my_state + 1
        return my_state
    return count_up
```

```
>>> counter = make_counter()
>>> counter()
# FILL ME IN
>>> counter()
# FILL ME IN
>>> counter()
# FILL ME IN
```

Aside: Counters and nonlocal/global

- Here is another implementation. What Would Python Print?

```
def make_counter():
    my_state = 0
    def count_up():
        my_state = my_state + 1
        return my_state
    return count_up
```

```
>>> counter = make_counter()
>>> counter()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 4, in count_up
UnboundLocalError: local variable
'my_state' referenced before assignment
```

Woah. What is going on here?

Due to language design decisions, Python does not let you re-bind variables that exist in parent frames (including the global frame). Same thing with “+=“

For a more detailed explanation straight from the Python3 docs, see this link.

Aside: Counters and nonlocal/global

- One way to fix this is to use the nonlocal keyword

```python
def make_counter_v3():
    my_state = 0
    def count_up():
        nonlocal my_state
        my_state = my_state + 1
        return my_state
    return count_up
```

```
>>> counter = make_counter()
>>> counter()
1
>>> counter()
2
```

`nonlocal my_state` tells Python that the `my_state` variable refers to a variable in a parent (non-local) frame, and any assignments should modify the variable in the OTHER frame.

Aside: Counters and nonlocal/global

- There’s an analogous keyword for global vars, `global`

```
a_global_var = 42
def fn():
    global a_global_var
    a_global_var = 9000

>>> print(f"Before fn(): {a_global_var}")
Before fn(): 42
>>> fn()
>>> print(f"After fn(): {a_global_var}")
After fn(): 9000
```

In Data C88C su24: nonlocal, global

- In this class (Data C88C Summer 2024), we will NOT be using the Python keywords: `nonlocal`, `global`
- For assignments/exams, we won't expect you to use nonlocal/global
- But, we will expect you to understand why the below errors

```python
def make_counter():
    my_state = 0
    def count_up():
        my_state = my_state + 1
        return my_state
    return count_up
```

```
>>> counter = make_counter()
>>> counter()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 4, in count_up
UnboundLocalError: local variable
'my_state' referenced before assignment
```

Aside: Counters and nonlocal/global

- (review) What Would Python Print?

```
a_global_var = 42
def fn2():
    a_global_var = 9000

>>> print(f"Before fn(): {a_global_var}")
42
>>> fn2()
>>> print(f"After fn(): {a_global_var}")
# FILL ME IN
```

Aside: Counters and nonlocal/global

- (review) What Would Python Print?

```
a_global_var = 42
def fn2():
    a_global_var = 9000

>>> print(f"Before fn(): {a_global_var}")
42
>>> fn2()
>>> print(f"After fn(): {a_global_var}")
42
```

This creates a new variable `a_global_var` in the fn2 frame, with value 9000. Notably, this `a_global_var` shadows the `a_global_var` at the global frame.
Visualization: Python Tutor

Computational Structures in Data Science

Abstract Data Types

Berkeley
UNIVERSITY OF CALIFORNIA

Abstract Data Type

- This is a different style of content than before.
- ADT's is more about a style of programming
- "Disciplined" programming, "best practices"

Abstract Data Type

- Uses pure functions to encapsulate some logic as part of a program.
- We rely on built-in types (int, str, list, etc) to build ADTs
- This is in contrast to object-oriented programming
  - Which is coming soon!

Creating Abstractions

- Compound values combine other values together
  - date: a year, a month, and a day
  - geographic position: latitude and longitude
  - a game board

- Data abstraction lets us manipulate compound values as units
- Isolate two parts of any program that uses data:
  - How data are represented (as parts)
  - How data are manipulated (as units)
- **Data abstraction**: A methodology by which functions enforce an abstraction barrier between *representation* and *use*

Why Abstract Data Types?

- In code, how do you represent a game board, a "course", a person, a student?
- One of the “arts” of well-written code is to build effective, useful abstractions that make solving problems easier
- Examples:
- Python: provides built-in data types like int, str, tuple, list, and dict. Developers take these “base” data types and build new data types on top of them to solve problems
  - Example: represent a 2-D matrix as a list of lists. Define matrix operations (matmult) in terms of list operations.

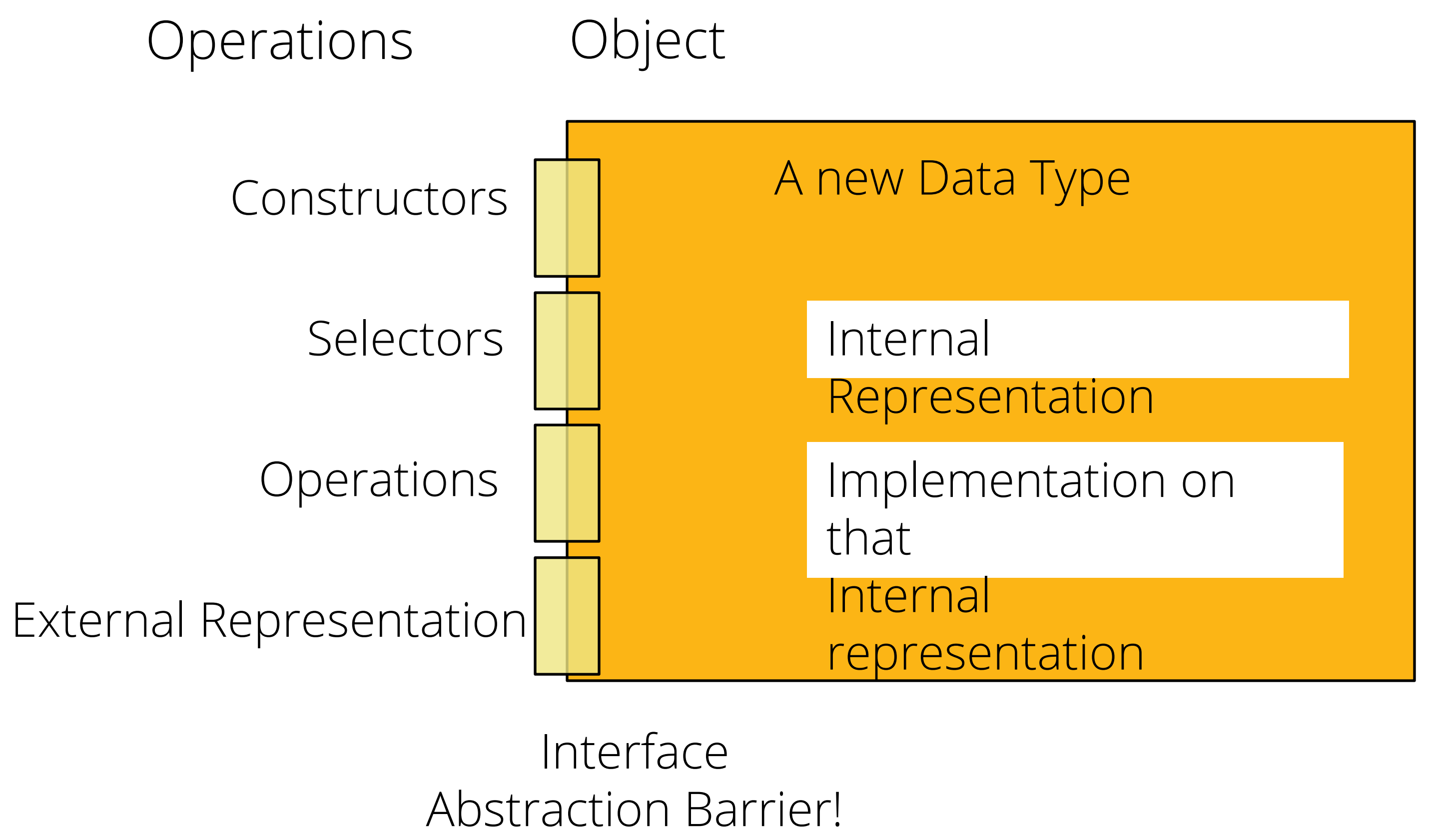
Why Abstract Data Types?

- numpy , a Python library, takes this idea to the extreme: it defines a (very) performant matrix data type. It is a cornerstone in many ML/DS projects (and a gold-star example of what open-source can achieve!)
- Pytorch, also a Python library. Defines the Tensor and nn.Module data types. With these two types, we can build and train state-of-the-art neural networks like ChatGPT.
  - (a simplification, but you'd be surprised at how close it is to the truth)

NumPy

PyTorch

Abstract Data Type



C.O.R.E concepts

Compute	Perform useful computations treating objects abstractly as whole values and operating on them.
Operations	Provide operations on the abstract components that allow ease of use – independent of concrete representation.
Representation	Constructors and selectors that provide an abstract interface to a concrete representation
Evaluation	Execution on a computing machine

Abstract Data Type

Abstraction Barrier

Example: the point ADT

Suppose we wanted to define a “2d point” data type. A 2d point has an x coordinate, and a y coordinate.

Question: in Python, how would you represent a 2d point?

Example: the point ADT

Suppose we wanted to define a “2d point” data type. A 2d point has an x coordinate, and a y coordinate.

(Answer) Let’s represent a 2d point as a list with two elements: [int x, int y]

Note: there are many ways you could have implemented this

Example: the point ADT

An “undisciplined” way of working with our “2d point” data type would be to work at the Python list level, writing code like this:

```
point_a = [1, 2]
point_b = [4, 5]

def distance_l2(p1, p2):
    # Calculates L2 distance between 2d points p1, p2
    return ((p1[0] - p2[0])**2 + (p2[1] - p2[1])**2)**0.5

>>> print(f"dist btwn point_a and point_b: {distance_l2(point_a,
point_b)}")
dist btwn point_a and point_b: 3.0
>>> print(f"x coord of point_a is: {point_a[0]}")
x coord of point_a is: 1
>>> print(f"y coord of point_a is: {point_a[1]}")
y coord of point_a is: 2
```

Example: the point ADT

- While it does work, the resulting code has the following issues:
- There is no abstraction in the `distance_l2()` function. It assumes that a point is a list [x, y], and does direct list indexing
- Aka "assumes the 2d point internal representation"

```python
point_a = [1, 2]
point_b = [4, 5]

def distance_l2(p1, p2):
    # Calculates L2 distance between 2d points p1, p2
    return ((p1[0] - p2[0])**2 + (p2[1] - p2[1])**2)**0.5
```

This code only works if p1, p2 are lists of the format [x, y]. A little brittle

Example: the point ADT

- What if we need to change the 2d point internal representation?
- Example: suppose we want to attach a “str color” to a point?

```
~~point_a = [1, 2]~~     point_a = [“red", 1, 2]
~~point_b = [4, 5]~~     point_b = [“blue", 4, 5]

def distance_l2(p1, p2):
    # Calculates L2 distance between 2d points p1, p2
    return ((p1[0] - p2[0])**2 + (p2[1] - p2[1])**2)**0.5
```

Now, this function will error! We now need to change (refactor) all code that uses our 2d point data type to adjust to the new internal representation

Example: the point ADT

- It may seem OK if it's just one function, but in larger software projects, there may be literally millions of lines of code to change…

```
~~point_a = [1, 2]~~
~~point_b = [4, 5]~~
```

```
point_a = [“red", 1, 2]
point_b = [“blue", 4, 5]

def distance_l2(p1, p2):
    # Calculates L2 distance between 2d points p1, p2
    return ((p1[0] - p2[0])**2 + (p2[1] - p2[1])**2)**0.5

def distance_l1(p1, p2):
    # ...
def norm_l2(p1, p2):
    # ...
def norm_l1(p1, p2):
    # ...
def sum_vals(p1, p2):
    # ...
def cosine_similarity(p1, p2):
    # ...
# ...
```

Oops, we've got our work cut out for us…
And, worse, this is tedious, manual, error-prone work…
…could we have planned better ahead to avoid this pain?

Example: the point ADT

- Idea: let’s implement `distance_l2()` in a more abstract, generic way. Notably, one that doesn’t assume the internal representation of the point data type.

```
~~point_a = [1, 2]~~    point_a = [“red", 1, 2]
~~point_b = [4, 5]~~    point_b = [“blue", 4, 5]

def distance_l2(p1, p2):
    # Calculates L2 distance between 2d points p1, p2
    return ((p1[0] - p2[0])**2 + (p2[1] - p2[1])**2)**0.5
```

These p1[0], p2[0] is really asking for “get me the x coordinate”

Similarly, p1[1], p2[1] is really asking for “get me the y coordinate”

Example: the point ADT

- Idea: let's implement `distance_l2()` in a more abstract, generic way. Notably, one that doesn't assume the internal representation of the point data type.

```
~~point_a = [1, 2]~~
~~point_b = [4, 5]~~
```

```
point_a = [“red", 1, 2]
point_b = [“blue", 4, 5]
```

```
def distance_l2_abstract(p1, p2):
    # Calculates L2 distance between 2d points p1, p2
    return ((get_x(p1) - get_x(p2))**2 + (get_y(p2) - get_y(p2))**2)**0.5
```

So, let's just ask via get_x()!

And, ask for y via get_y()

Example: the point ADT

- Finally, let's define the constructor and selector functions to fully spec out our point ADT

```
~~point_a = [1, 2]~~
~~point_b = [4, 5]~~
```

```
point_a = [“red", 1, 2]
point_b = [“blue", 4, 5]
```

```
def distance_l2_abstract(p1, p2):
    # Calculates L2 distance between 2d points p1, p2
    return ((get_x(p1) - get_x(p2))**2 + (get_y(p2) - get_y(p2))**2)**0.5
```

```
# constructor
def create_point(x, y, color):
    return [color, x, y]
```

```
# selectors
def get_x(point):
    return point[1]
def get_y(point):
    return point[2]
def get_color(point):
    return point[0]
```

Example: a Point ADT

```
# constructor
def create_point(x, y, color):
    return [color, x, y]

# selectors
def get_x(point):
    return point[1]
def get_y(point):
    return point[2]
def get_color(point):
    return point[0]
```

This is the ADT. (“Under the hood”, “below the abstraction barrier”, etc).
It’s allowed to know details about the **internal representation** of the data type, eg “a Point is implemented as a list of three elements”

```
# Operators
def distance_l2_abstract(p1, p2):
    # Calculates L2 distance between 2d points p1, p2
    return ((get_x(p1) - get_x(p2))**2 + (get_y(p2) - get_y(p2))**2)**0.5
```

Example: a Point ADT

```python
# constructor
def create_point(x, y, color):
    return [color, x, y]

# selectors
def get_x(point):
    return point[1]
def get_y(point):
    return point[2]
def get_color(point):
    return point[0]
```

These are the **operations** that are built on top of the abstractions defined by the ADT.
They are NOT allowed to know details about the internal representation.
Instead, they should only use the ADT's "public-facing API/spec", aka the **constructors and selectors**

This is the "abstraction barrier".
Don't cross the boundary!

```python
# Operators
def distance_l2_abstract(p1, p2):
    # Calculates L2 distance between 2d points p1, p2
    return ((get_x(p1) - get_x(p2))**2 + (get_y(p2) - get_y(p2))**2)**0.5
```

Lists: an ADT too!

- Lists
  - Constructors:
    - list(...)
    - [<exps>,...]
    - [<exp> for <var> in <list> [if <exp>]]
  - Selectors: <list> [<index or slice>]
  - Operations: in, not in, +, *, len, min, max

Question: What is the internal representation of a Python list?

Lists: an ADT too!

- Lists
  - Constructors:
    - list(...)
    - [<exps>,...]
    - [<exp> for <var> in <list> [if <exp>]]
  - Selectors: <list> [<index or slice>]
  - Operations: in, not in, +, *, len, min, max

Question: What is the internal representation of a Python list?

Answer: It depends (ha). But if you're running CPython (which you probably are), it's likely backed by a C++ implementation…which is compiled to assembly…which is translated to machine code…

Observation: abstraction is a somewhat subjective + relative term. For instance, our Point ADT uses Lists as its internal representation. But, at a lower abstraction level, you can think of Lists as an ADT as well.

Creating an Abstract Data Type

- Constructors & Selectors
  - This basically is the ADT, and encompasses the **internal representation** of the data type.
- Operations
  - Additional functionality **built on top of** the ADT constructions + selectors.
  - **Crucially, it must not make assumptions about the internal representation of the data type,**

Defining The Abstraction Barrier

- An **abstraction barrier violation** occurs when a part of the program that can use the "higher level" functions uses "lower level" ones instead
  - At either layer of abstraction
  - e.g. Should your function be aware of the implementation?
    - Be consistent!
- Abstraction barriers make programs easier to get right, maintain, and modify
  - Fewer changes when representation changes

A Layered Design Process – Bottom Up vs Top Down

- Start with "What do you want to do?"
- Build the application based entirely on the ADT interface
  - Focus first on Operations, then Constructors and Selectors
  - Do not implement them! Your program won't work.
  - You want to capture the "user's" point of view
- Build the operations in ADT on Constructors and Selectors
  - Not the implementation representation
  - This is the end of the abstraction barrier.
- Build the constructors and selectors on some concrete representation

Extended Demo: Tic Tac Toe and Phone Book

- See the companion notebook.
- Download the file "ipynb"
  - Go to https://datahub.berkeley.edu
  - Log in, then select "Upload"

Lecture overview. Any questions?

- Mutable Functions (finish off slides we didn't get to yesterday)
- Abstract Data Types ("ADT")
  - A central idea for Project01 ("Maps")