CS234

MODULE 2 – ADTS

• What is a data type?
• What is abstraction?
• What is an abstract data type?
• Python Iterators
Computer Data

Computers store data in binary (base 2 number system)
• Usually grouped into 8 bits (short for binary digits)
• 8 bits is called a “byte”

Q: What does the following represent?
   0000 1000 0100 0111 1110 1001 1111 1010

A: It could be an integer. Or a float. Or part of a string. Or …
• We need some way of knowing. Some way like a…
Data Type

A data type is a way of assigning meaning to 0s and 1s

- A set of possible values and the corresponding binary pattern
  - E.g. The set of non-negative integers
- A collection of operations for manipulating the values
  - E.g. how to add, subtract, multiply, divide the non-negative integers

Typically a programming language will provide a few different data types to work with.
Data Types

We can divide data types into two broad groups

Simple (or scalar)

• Represents a single value
• E.g. integer, floating point, boolean, character

Complex (or compound)

• Represents a value with multiple components (possibly an arbitrary number of them)
• E.g. list, string
Data Types

Another way to categorize data types is by where they came from

Primitive (or built-in)
- Part of the language itself (int, float, string)
- Usually fairly general

User-Defined
- Created by the user as needed (bank account, goblin)
- Can be quite specific
What Abstraction Means

Abstraction: Separating the properties of an object and focusing on the relevant ones (ignoring the rest)

Or

Big Picture: Worry about what, not how
Abstraction

Abstraction is how people deal with complexity

Q: Can you picture Waterloo all at once?
A: Probably not (maybe it’s just me?)

Need a mental model of the city to get around.

(Or, Google Maps, but a map is another abstraction)
Computers and Abstraction

When you post a meme, an awful lot is happening

• Millions of calculations
• Electrons zipping through transistors left right and center
• Network hardware all over the world is receiving, processing, and forwarding data packets

Do you know how any of that works? Maybe, but you don’t have to.
Computers and Abstraction

This applies to programming, too

Q: **What** is `math.cos(0)`?
A: 1.0

Do you know **how** Python computes cosine?
• Nah

Do you need to?
• Nah
Kinds of Abstractions

Functional Abstraction:
• We can use a method / function without knowing its implementation details
• E.g. we don’t need to know how math.cos works

Data Abstraction:
• We don’t need to know how Python represents a type in binary to use it
• E.g. 2 + -3 ⇒ -1, regardless of implementation details
Abstract Data Type (ADT)

• A user-defined type where all you document is
  • A set of values
  • A set of well-defined operations
    • The operations might involve other types, too

• Abstract because:
  • Don’t need to know how the implementation works to use the operations
Collection ADTs

A collection ADT is an ADT designed to store an arbitrary number of items.

In Python you’ve used three of these already:

*List*, stores an arbitrary number of values

*Dictionary*, stores an arbitrary number of key-value pairs

*String*, stores an arbitrary number of characters.
String ADT

- Lots of methods
  - `dir(str)`
  - This is the interface
  - You don’t know the implementation, and don’t need to
  - You can view it as a black box
Information Hiding

Implementation details are hidden from the user.

Advantages:

• Security
  • The User cannot access the characters in S directly. This ensures that strings are immutable (Python relies on this)

• Flexibility
  • If the implementation changes, the User doesn’t need to know (their code can stay the same)
Dictionary ADT

Some major operations

```
d[key]                # retrieves the value associated with key
d[key] = value       # associates key with value
key in d             # checks if d contains key
```
Dictionary ADT

In previous courses you have seen two different ways to implement an ADT:
• An Association List
• A Binary Search Tree

Because it’s **abstract** you don’t need to know which one the Python dictionary is.

(It’s neither)
Operations

An ADTs methods can usually be grouped into broad categories

- **Constructor**: A way of making a new instance of this ADT
  - Sometimes out of existing ones, e.g. String + String ⇒ New String

- **Accessor**: A way to retrieve a value from the ADT
  - E.g. String[i]

- **Mutator**: A way of changing the state of the ADT
  - Adding a value, removing one, changing an existing one

- **Iterator**: A way of traversing (iterating over) the ADT
  - Lets a for loop know how to do its business
Defining an Operation

You need to describe what each operation does

• What parameters does it have?
• What does it return (if anything)?
• What are the preconditions?
• How does it change the state of the ADT? (postconditions)
Precondition?

A precondition is something that must be true before the operation begins.
  • This includes parameter values, but also the state of the ADT

If the precondition is not met, the behaviour is undefined.

Users prefer few preconditions.
Implementers prefer many preconditions.
Precondition Examples

Example 1:
• Let L be a List
• \texttt{L.pop(i)} has the precondition that \texttt{i} is a valid index in \texttt{L}.
  • If this is not met, it throws an index out of range error

Example 2:
• \texttt{math.sqrt(x)} requires \texttt{x} \geq 0
Postcondition?

Something that will be true after the operation is completed.

Usually this will be a change to the state of the ADT.

It also includes the returned value, if there is anything important to say about it.
Postcondition Examples

Example 1:
• Let L be a List
• After L.pop(i), the element at index i will be removed from L and returned

Example 2:
• Let D be a Dictionary
• After D[k] = v, key k will be associated with value v (replacing the old value if k was already in D)
This sounds familiar

Yes, Preconditions and Postconditions are more or less “requires” and “effects” from CS116/136

• The old style guide even used “PRE” and “POST” as the labels!

• This was changed because...I don’t know

Whatever you call them, they are important!

There’s a word for an undocumented ADT: Trash
Something New

If the user tries to violate a precondition (something the function “requires”) it should raise an error just like a built-in.

One way to do this is to assert the preconditions.

```python
def my_function(x):
    '''Requires: x is not 0'''
    assert x != 0, "X must not be 0"
```
Example: Date ADT

A Date represents a single day on the (Gregorian) calendar.

<table>
<thead>
<tr>
<th>Operation(s)</th>
<th>Kind of Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date(year,month,day)</td>
<td>Constructor</td>
</tr>
<tr>
<td>day(), month(), year()</td>
<td>Accessor</td>
</tr>
<tr>
<td>monthName(), isLeapYear()</td>
<td>Accessor</td>
</tr>
<tr>
<td>toString(), compare(other)</td>
<td>Accessor</td>
</tr>
<tr>
<td>daysBetween(other)</td>
<td>Accessor</td>
</tr>
<tr>
<td>advance(numDays)</td>
<td>Mutator</td>
</tr>
</tbody>
</table>
Example: Date ADT

We can implement this ADT in Python. It’s not a collection so there’s not much thought needed

• OK, daysBetween probably involves some math
Collection ADTs

The Date ADT is simple. While it can hold an arbitrary number of values, it only has 3 integral values at a time.

In a Collection, there are an arbitrary number of values. We will need to hold these in some kind of data structure.

Or, maybe in one or more nested ADTs? Sure!
Example: Stack

A Stack holds an arbitrary number of values. You can only add or remove from the top of the stack, though.

<table>
<thead>
<tr>
<th>Operation(s)</th>
<th>Kind of Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stack()</td>
<td>Constructor</td>
</tr>
<tr>
<td>length()</td>
<td>Accessor</td>
</tr>
<tr>
<td>top()</td>
<td>Accessor</td>
</tr>
<tr>
<td>pop()</td>
<td>Mutator</td>
</tr>
<tr>
<td>clear()</td>
<td>Mutator</td>
</tr>
<tr>
<td>items()</td>
<td>Iterator (<strong>new</strong>*)</td>
</tr>
</tbody>
</table>
Iterators

Most of the stack is boring (it’s just a list with some restrictions on accessing, inserting, and popping).

The Iterator method is new though.
My First Iterator

Iterators are used for traversing collections. You’ve used them before!

for v in my_list:
    print(v)

A for loop uses an Iterator object to figure out what values v should take on each iteration, and how many iterations there should be.
Defining a Stack Iterator Class

class StackIterator:
    def __init__(self, items):
        self._items = items
        self._pos = 0
    def __iter__(self):
        return self
    def __next__(self):
        if self._pos < len(self._items):
            item = self._items[self._pos]
            self._pos += 1
            return item
        else: raise StopIteration
Making an Iterator Object

```python
class Stack:
    ...
    def __iter__(self):
        return StackIterator(self._items)
```

An object’s `__iter__` magic method returns an Iterator Object.

The Iterator’s `__next__` magic method returns the next value, or raises the StopIteration exception.
Iterators and Iterable Objects

Anything class that implements the \_\_iter\_\_ magic method is “Iterable.”

- \_\_iter\_\_ must return an Iterator
- An Iterator must implement \_\_iter\_\_

Iterators are used by
- For loops with the “in” syntax
- list(), map(), filter(), etc.