

CS 134 Lecture 29:  
Special Methods & Linked Lists

# Announcements & Logistics

- **HW 9** due Monday @ 10 pm on GLOW
  - Short: 6 questions for practice on OOP concepts
- **Lab 9 Boggle**: two-week lab now in progress
  - **Part 1** auto-tester feedback will be returned today
  - You can fix anything broken before turning in Part 2
  - **Part 2** due May 1/2 (handout posted)
  - Part 2 also has a **prelab!**
    - Asks you to draw out the Boggle game logic

**Do You Have Any Questions?**

# Last Time

- Finished implementation of **Tic Tac Toe game**
  - (Fun?) Application of object-oriented design and inheritance
  - A little exposure to software design
- Designed to help with the **Boggle lab**

# Today's Plan

- Discuss special methods, their purpose and how to that call them
- Build a **recursive list class**
  - Our own implementation of **list**!
  - Preview of the fun world of design and implementation of data structures
- Learn how to implement several **special methods** which let us utilize built-in operators in Python for user-defined types

# Python's Built-in list Class

- A class with methods (that someone else implemented)
- `pydoc3 list`
- Let's implement our own list class with similar functionality

Notice the double underscores: these are special methods

```
Help on class list in module builtins:
```

```
class list(object)
| list(iterable=(), /)
|
| Built-in mutable sequence.
|
| If no argument is given, the constructor creates a new empty list.
| The argument must be an iterable if specified.
|
| Methods defined here:
|
| __add__(self, value, /)
|     Return self+value.
|
| __contains__(self, key, /)
|     Return key in self.
|
| __delitem__(self, key, /)
|     Delete self[key].
|
| __eq__(self, value, /)
|     Return self==value.
|
| __ge__(self, value, /)
|     Return self>=value.
|
| __getattr__(self, name, /)
|     Return getattr(self, name).
|
| __getitem__(...)
|     x.__getitem__(y) <==> x[y]
|
| __gt__(self, value, /)
```

Special Methods/Magic Methods

# Special Methods

- Start and end with `__` (double underscore)
  - Called **magic methods** (or informally dunder methods)
- Often not called explicitly using dot notation and called by other means
- What special methods have we already used seen/used so far?
- **`__init__(self, val)`**
  - When is it called?
    - Automatically when we **create** an instance (object) of the class
    - Can also be invoked as **`obj.__init__(val)`** (where **`obj`** is an instance of the class)

# Special Methods

- **`__str__(self)`**

- When is it called?

- When we ***print*** an instance of the class using `print(obj)`

- Also called whenever we call `str` function on it: `str(obj)`

- Can also be invoked as `obj.__str__()`

- **`__repr__(self)`**

- Also returns a string but its format is very specific (can be used to recreate the object of the class)

- Useful for debugging

- Don't worry about any more specifics for this method for CS134



# Special Methods for Operators

- We can use mathematical and logical operators such as `==/+` to compare/add two objects of a class by defining the corresponding special method
- Example of polymorphism (using a single method or operator for different uses)

- `__eq__ (self, other):`

`x == y`

- `__ne__ (self, other):`

`x != y`

- `__lt__ (self, other):`

`x < y`

- `__gt__ (self, other):`

`x > y`

- `__add__ (self, other) :`

`x + y`

- `__sub__ (self, other):`

`x - y`

- `__mul__ (self, other):`

`x * y`

`__add__`: why we can concatenate sequences with `+` as well as add ints with `+`

- There are many others!

# Special Method: `__len__`

- `__len__(self)`
  - Called when we use the built-in function `len()` in Python on an object `obj` of the class: `len(obj)`
  - We can call `len()` function on any object whose class has the `__len__()` special method implemented
  - All built-in collection data types we saw (string, list, range, tuple, set, dictionaries) have this special method implemented
  - This is why we are able to call `len` on them
- What is an example of a built-in type that we can't call `len` on?
  - `int, float, Bool, None`

# Python's Built-in list Class

- A class with methods (that someone else implemented)
- **pydoc3 list**
- Let's implement our own list class with similar functionality

Other sequence specific methods:  
**`__getitem__`**

Help on class list in module builtins:

```
class list(object)
| list(iterable=(), /)
|
| Built-in mutable sequence.
|
| If no argument is given, the constructor creates a new empty list.
| The argument must be an iterable if specified.
|
| Methods defined here:
|
| __add__(self, value, /)
|     Return self+value.
|
| __contains__(self, key, /)
|     Return key in self.
|
| __delitem__(self, key, /)
|     Delete self[key].
|
| __eq__(self, value, /)
|     Return self==value.
|
| __ge__(self, value, /)
|     Return self>=value.
|
| __getattr__(self, name, /)
|     Return getattr(self, name).
|
| __getitem__(...)
|     x.__getitem__(y) <==> x[y]
|
| __gt__(self, value, /)
```

# Other Special Methods for Sequences

- What other sequence operators have we used in this class?
- They each have a special method that is called whenever they are used
  - **Get** an item at an index a sequence using `[ ]:` calls `__getitem__`
    - e.g., `word_lst[2]` implicitly calls `word_lst.__getitem__(2)`
  - **Set** an item at an index to another `val` using `[ ]:` calls `__setitem__`
    - e.g., `word_lst[0] = "hello"` implicitly calls `word_lst.__setitem__(0, "hello")`

# **in** Operator: `__contains__`

- `__contains__(self, val)`
  - When we say `if elem in seq` in Python:
    - Python calls the `__contains__` special method on `seq`
    - That is, `seq.__contains__(elem)`
- If we want the `in` operator to work for the objects of our class, we can do so by implementing the `__contains__` special method

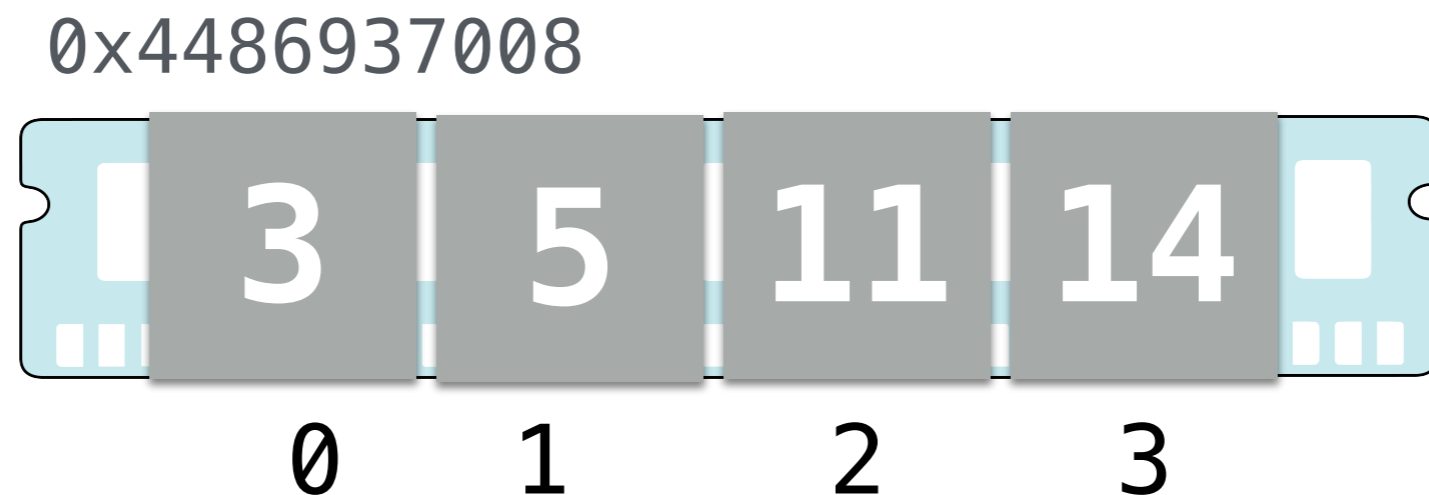
# Building Our Own Sequence Type

# How to Store a Sequence

- A sequence is just an **ordered collection** of values
  - Can query for the 0th, 1st, 2nd item and so on..
- A sequence may be mutable or immutable
- Let's think about how we can design such a sequence
  - How to store these ordered values?
- Let's look at two options

# Array: Contiguous Sequence

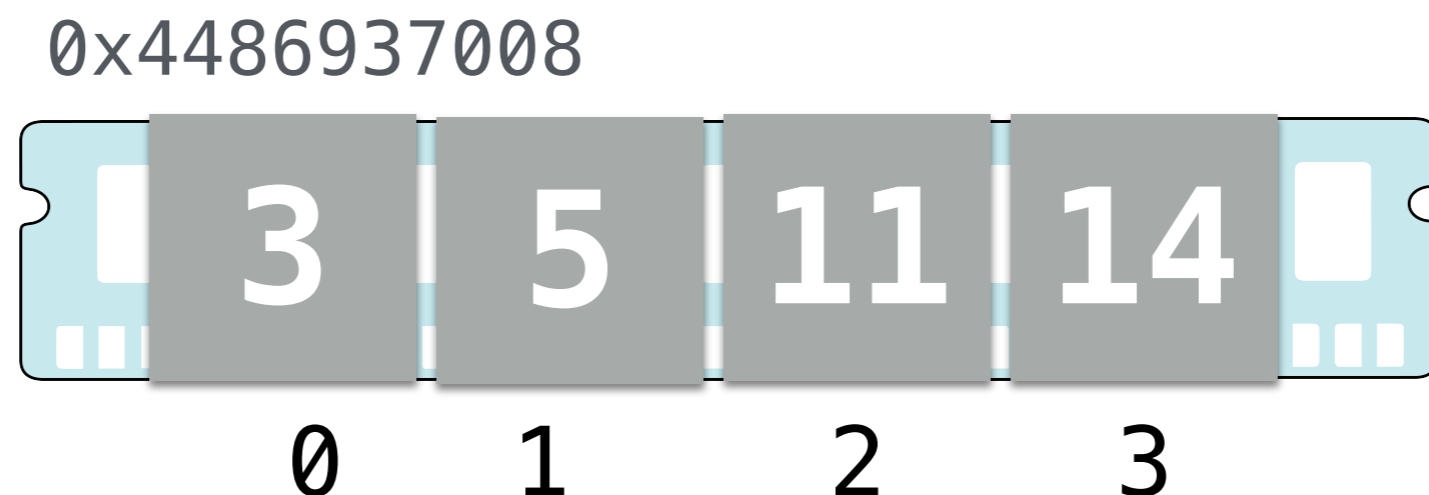
- Option 1: Just store the items contiguously in memory
- Such a sequence is called **an array** in computer science
- To access a **item**, just need to know where the array starts and the **index** of item in array
- Great for static sequences!





# Array: Contiguous Sequence

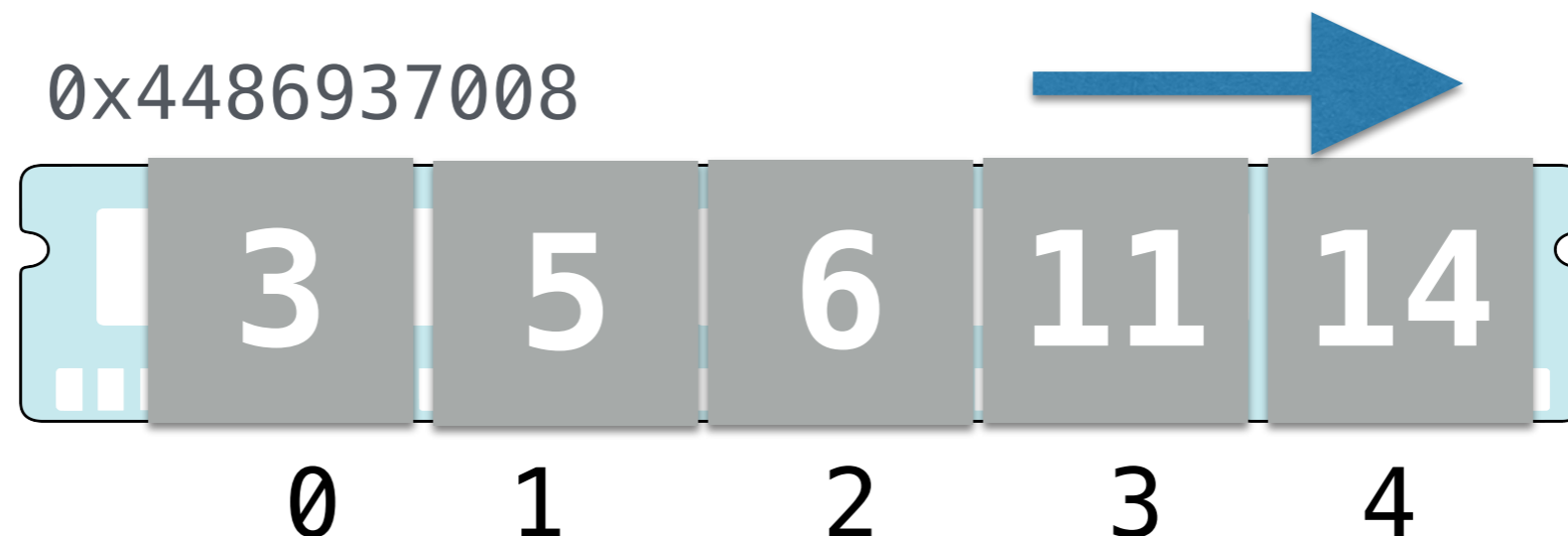
- Option 1: Just store the items contiguously in memory
- Such a sequence is called **an array** in computer science
- To access a **item**, just need to know where the array starts and the **index** of item in array
- Suppose we want to create a dynamic sequence
  - Want to be able to insert: e.g. want to insert 6 between 5 and 11



6

# Array: Contiguous Sequence

- Option 1: Just store the items contiguously in memory
- Such a sequence is called **an array** in computer science
- To access a **item**, just need to know where the array starts and the **index** of item in array
- Suppose we want to create a dynamic sequence
  - Want to be able to insert: e.g. want to insert 6 between 5 and 11
  - Need to move everything over by one to make space

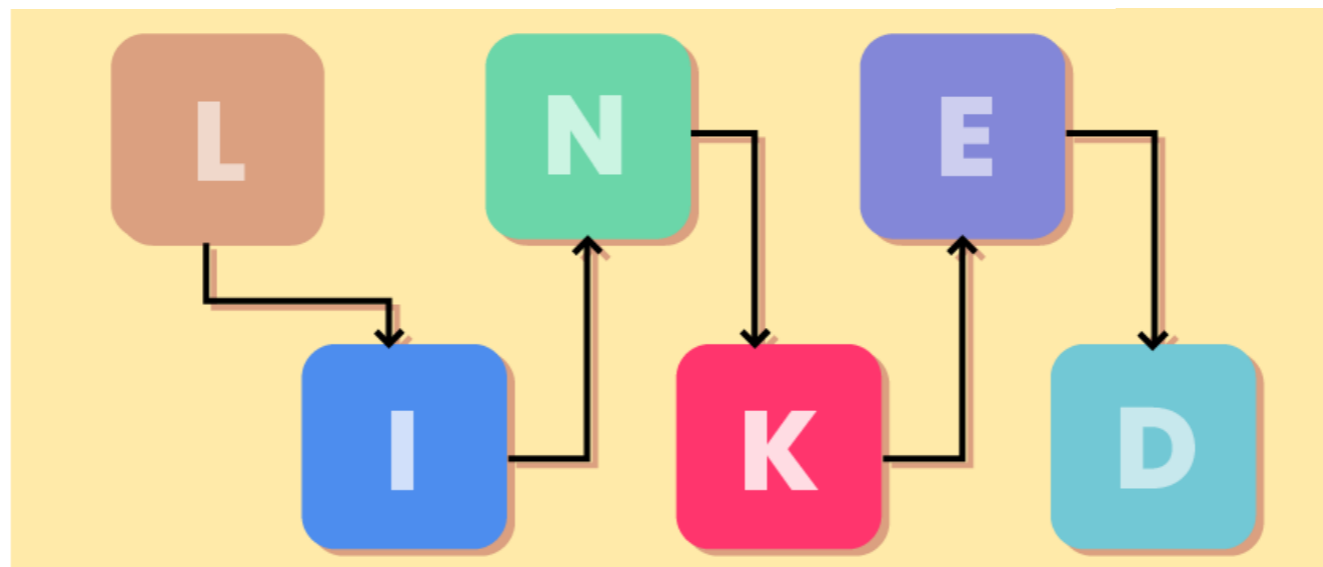


# Insert Efficiently?

- If our array is made up of millions of items and we need to insert a lot:
  - Expensive to maintain ordering in an array
  - Maybe we need a different way to store items
- All we care about is that items are in order:
  - Each item has a item before it or after it in the sequence
  - Knowing this is sufficient to recover the total ordering, why?

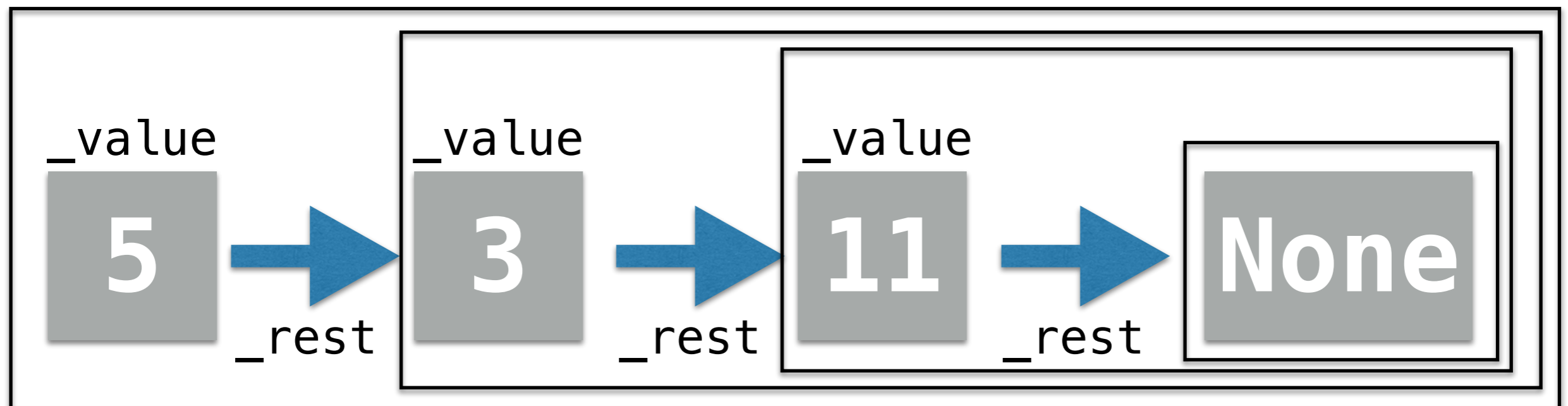
# Linked List

- Another way to design an ordered mutable sequence:
  - A nested *chain* of values, or a **linked list**
  - Each value has something after it: the rest of the sequence
- Must have a last item for a finite sequence
  - To signify last item it's next value should be **nothing**
  - What is a good type to represent nothing in Python?



# Our Own Class `LinkedList`

- Attributes:
  - `_value`, `_rest`
- **Recursive class:**
  - `_rest` points to another instance of the **same class**
  - Any instance of a class that is created by using another instance of the class is a **recursive class**



Next Time: Code for Linked List