LECTURE 5: ABC INTERFACES

ANDREY.BOCHARNIKOV@GMAIL.COM

TELEGRAM: @RICKO_X

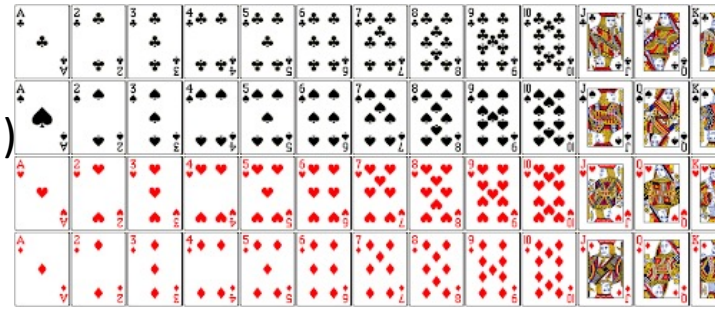
ABC INTERFACES

- Protocols vs abc interfaces
- ABC:
  - As superclass
  - Check for conformance to ABC interface
  - Mechanism to register conformant interface without subclassing
  - Recognizing that class conforms the interfaces without sublcassing and registering

PROTOCOLS AND DUCK TYPING

- The base sequence protocol in Python entails just the __len__ and __getitem__ methods

```
import collections
Card = collections.namedtuple('Card', ['rank', 'suit'])

class FrenchDeck:
    ranks = [str(n) for n in range(2, 11)] + list('JQKA')
    suits = 'spades diamonds clubs hearts'.split()

    def __init__(self):
        self._cards = [Card(rank, suit) for suit in self.suits
                                        for rank in self.ranks]

    def __len__(self):
        return len(self._cards)

    def __getitem__(self, position):
        return self._cards[position]
```



Don't check whether it is-a duck: check whether it quacks-like-a duck, walks-like-a duck, etc, etc, depending on exactly what subset of duck-like behavior you need to play your language-games with

Alex Martelli

PROTOCOLS

- Every class has an interface: set of public attributes (methods, data attributes) implemented or inherited including special methods, like __getitem__ or __add__.
- protected and private attributes are not part of an interface
  - Protected – naming convension (the single leading underscore)
  - Both easily accessed
- Protocols – informal interfaces

```
class Vector2d:
    typecode = 'd'

    def __init__(self, x, y):
        self.x = float(x)
        self.y = float(y)

    def __iter__(self):
        return (i for i in (self.x, self.y))
```

x, y as public attributes

```
class Vector2d:
    typecode = 'd'
    def __init__(self, x, y):
        self.__x = float(x)
        self.__y = float(y)

    @property
    def x(self):
        return self.__x

    @property
    def y(self):
        return self.__y

    def __iter__(self):
        return (i for i in (self.x, self.y))
```

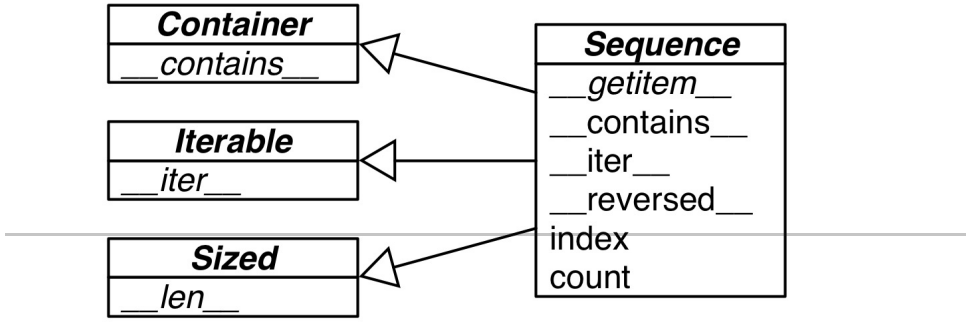
x, y as properties

INTERFACES

- "Interface - subset of an object's public methods that enable it to play a specific role in the system"
  - "File-like object" (https://docs.python.org/3/library/io.html)
  - "An iterable" (https://docs.python.org/3/glossary.html)
  - X-like object = X Protocol = X interface
    - "an object conforming to the buffer interfaces" => "bytes-like object"

SEQUENCES

- Foo doesn't inherit from abc.Sequence
  - Sequence requires __getitem__, __contains__, __iter__, __
- Foo implements only a signle method __getitem__
- Fallbacks
  - if no __iter__, iterate with __getitem__ starting with 0
  - Operator "***in***" works wihtout __contains__ method (full scan)



Sequence ABC and related abstract classes from collections.abc:
https://docs.python.org/3/library/collections.abc.html

```
class Foo:
    def __getitem__(self, pos):
        return range(0, 30, 10)[pos]
    # no __len__, __iter__ methods
```

MONKEY PATCHING

- FrechDeck implements only immutable sequence protocol and cannot be shuffled
- Mutable sequences must provide a __setitem__ method
- Monkey patching: changing a class at run time, without touching the source code
- Cons: the code that does the actual patching is very tightly coupled with the program to be patched, often handling private and undocumented parts.
- protocols are dynamic:
  - Shuffle needs the object to implement part of the mutable sequence protocol
  - Even if the object was "born" without the necessary methods and they were somehow acquired later (e,g, with monkey patching)

```
from random import shuffle

l = list(range(10))
shuffle(l)
print(l)
```

```
[5, 2, 9, 7, 8, 3, 1, 4, 0, 6]
```

DUCK TYPING

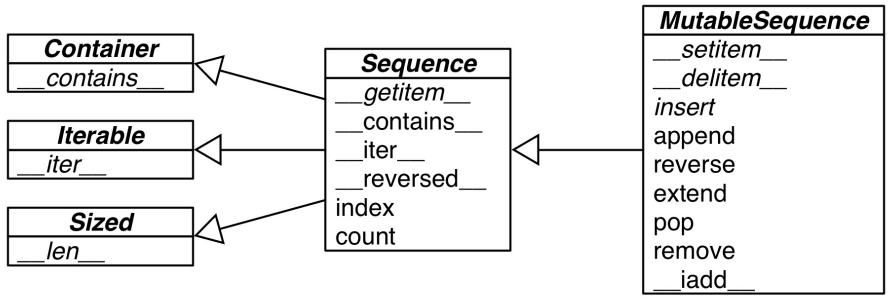
- "duck typing“ - ignoring an object’s actual type, focusing instead on ensuring that the object implements the method names, signatures, and semantics, required for its intended use.
- avoiding the use of ***isinstance*** to check the object’s type. And especially ***type(foo) is bar***
- Some methods may have the same name, but different meaning. Here duck typing works not well

```
class Artist:
    def draw(self): ... # paint

class Gunslinger:
    def draw(self): ... # grab

class Lottery:
    def draw(self): ... # pull
```

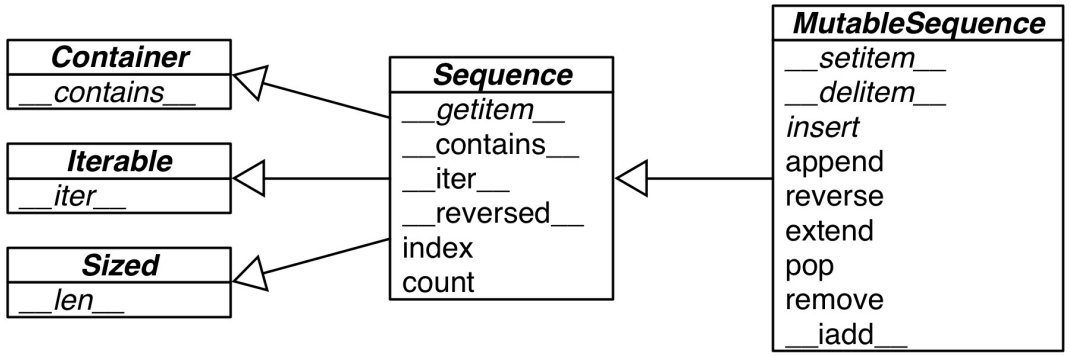
GOOSE TYPING

- Use isinstance(obj, cls) only when ***cls*** is an Abstract Base Class (in other words, cls's metaclass is abc.ABCMeta)
- **register** allows declaring a "virtual" subclass
  - registered class must meet method name and signature requirements
- This breaks coupling
- Sometimes register is not needed

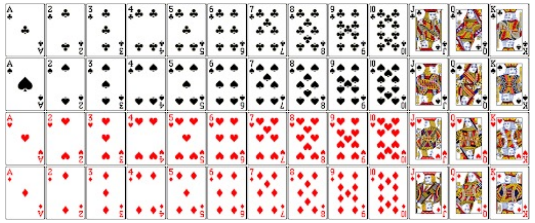
```
class Struggle:
    def __len__(self):
        return 23 ...

from collections import abc
isinstance(Struggle(), abc.Sized)
```



- MutableSequence requires implementing missing methods (__delitem__, insert)
- Not all methods are abstract. There are ready-to-use methods:
  - Sequence: __contains__, __iter__, __reversed__, index, and count
  - MutableSequence: append, reverse, extend, pop, remove, and __iadd__
- Override methods for better performance

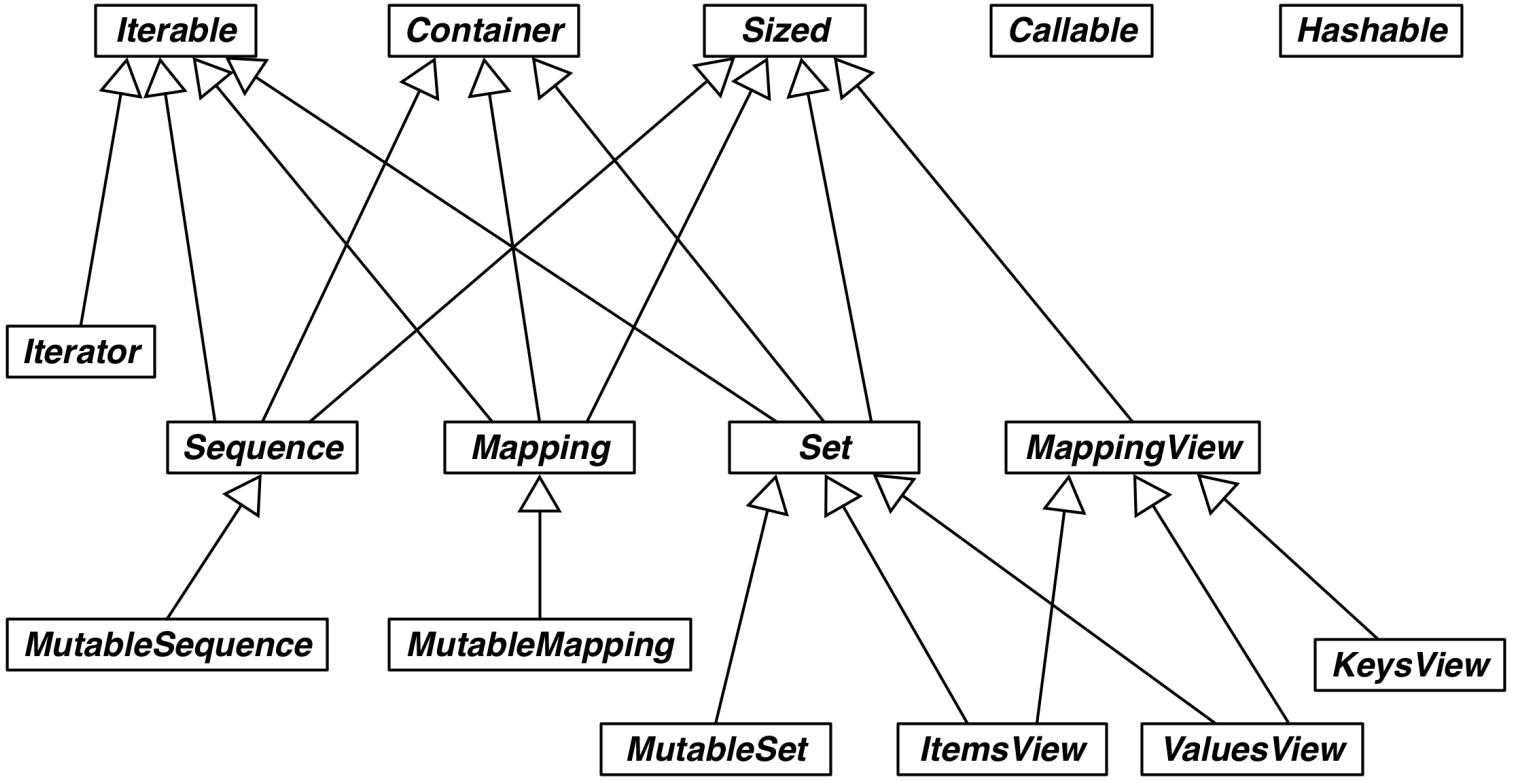



```
class FrenchDeck2(collections.MutableSequence):
    ranks = [str(n) for n in range(2, 11)] + list('JQKA')
    suits = 'spades diamonds clubs hearts'.split()

    def __init__(self):
        self._cards = [Card(rank, suit) for suit in self.suits
                                        for rank in self.ranks]

    def __len__(self):
        return len(self._cards)

    def __getitem__(self, position):
        return self._cards[position]

    def __setitem__(self, position, value):
        self._cards[position] = value

    def __delitem__(self, position):  # <-- MutableSequence
        del self._cards[position]

    def insert(self, position, value): # <-- MutableSequence
        self._cards.insert(position, value)
```

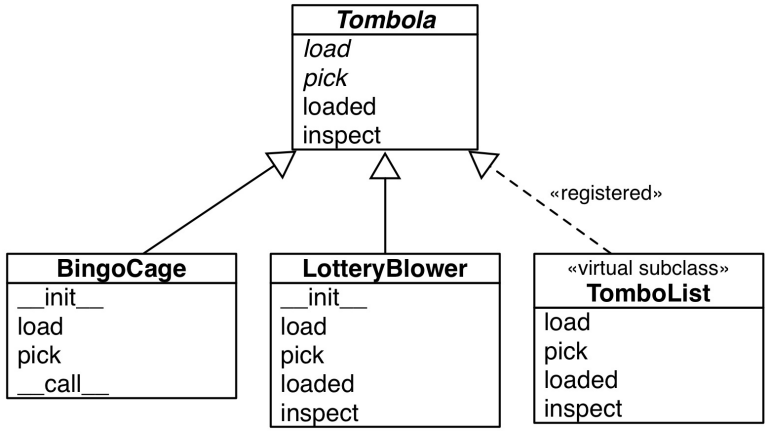


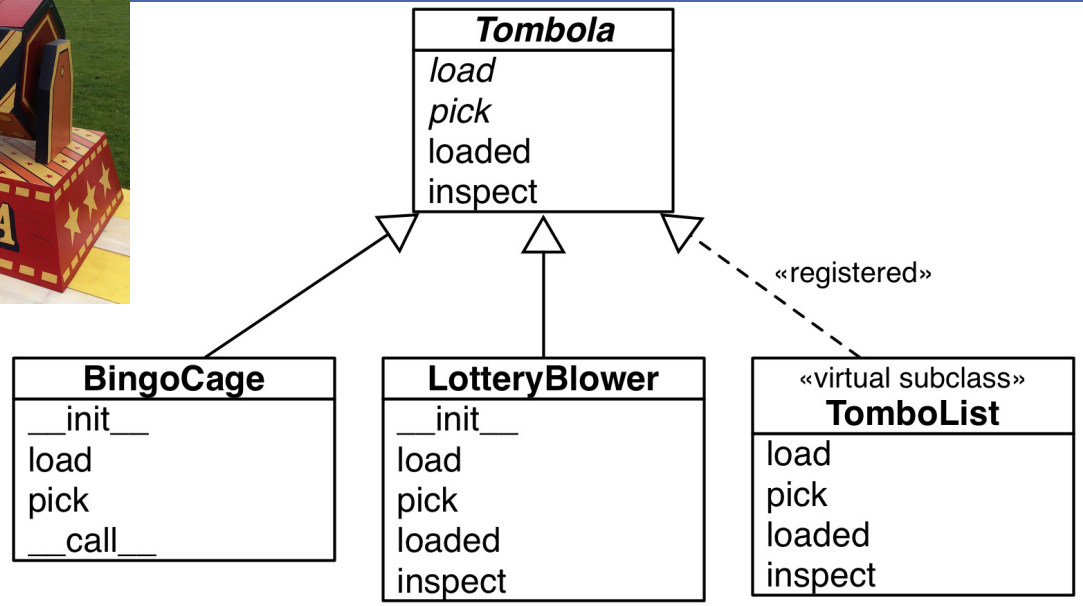
ABCS IN COLLECTIONS.ABC

- 2 standard modules:
  - collections.abc
  - Abc
- Iterable, Container and Sized – collections
- Sequence, Mapping and Set – immutable collections
- MappingView - objects returned from the mapping methods (like items, keys, values)
- Iterator - iterators



https://docs.python.org/3/library/collections.abc.html

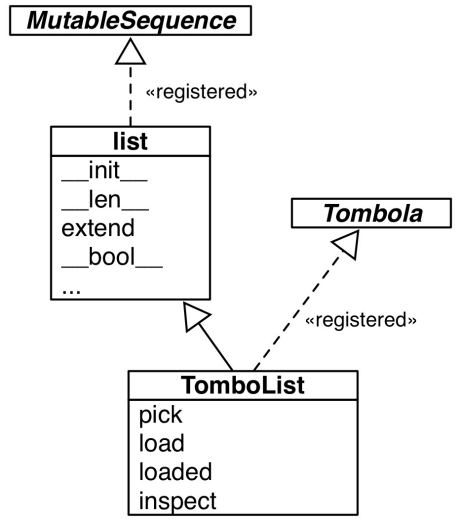




- Load – put itemsinto the container
- Pick – remove one item at random, returning it
- Loaded – if contains at least one item
- Inspect – return a tuple for items currently in the container

VIRTUAL SUBCLASS

- The registered class then be- comes a virtual subclass of the ABC, and will be recognized as such by functions like issubclass and isinstance, but it will not inherit any methods or attributes from the ABC.
- No runtime checks(!)



THAT'S IT

- Questions?
- Discussing the 1st assignment: 24 Apr 12:30
- Next Lecture: 28 Apr 14:30