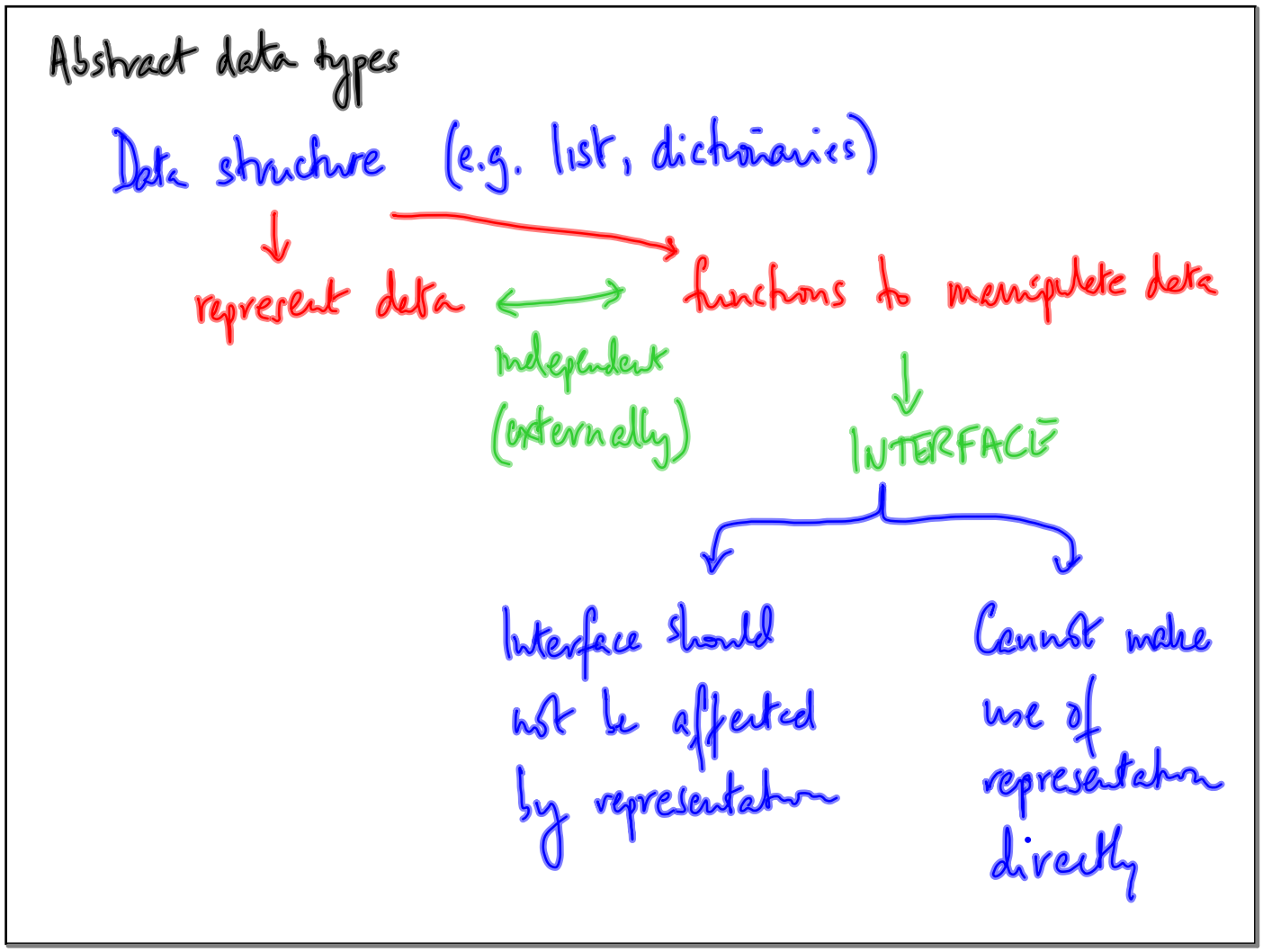
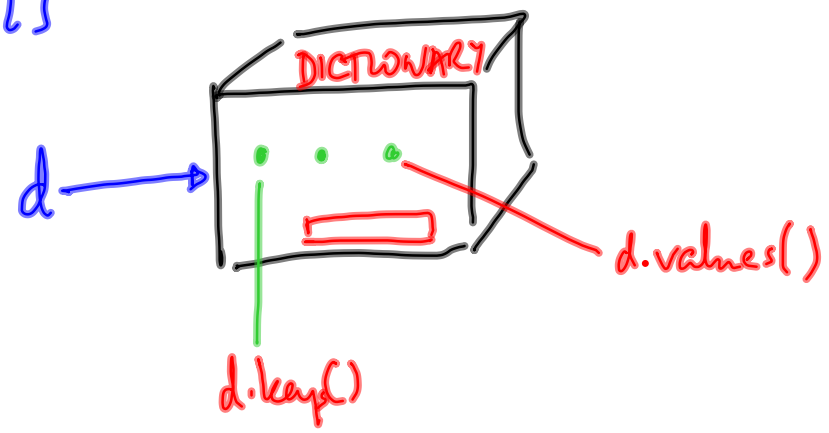

Abstract data types
Data structure (e.g. list, dictionaries)
represent data
functions to manipulate data
independent (externally)
INTERFACE
Interface should not be affected by representation
Cannot make use of representation directly

Encapsulation

One approach is "object oriented" programming

A way of separating implementation from interface

```
d = {}
```

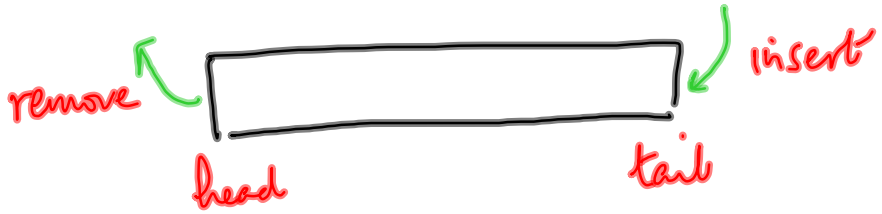


Two step process

Define the representation & functions — Template / CLASS

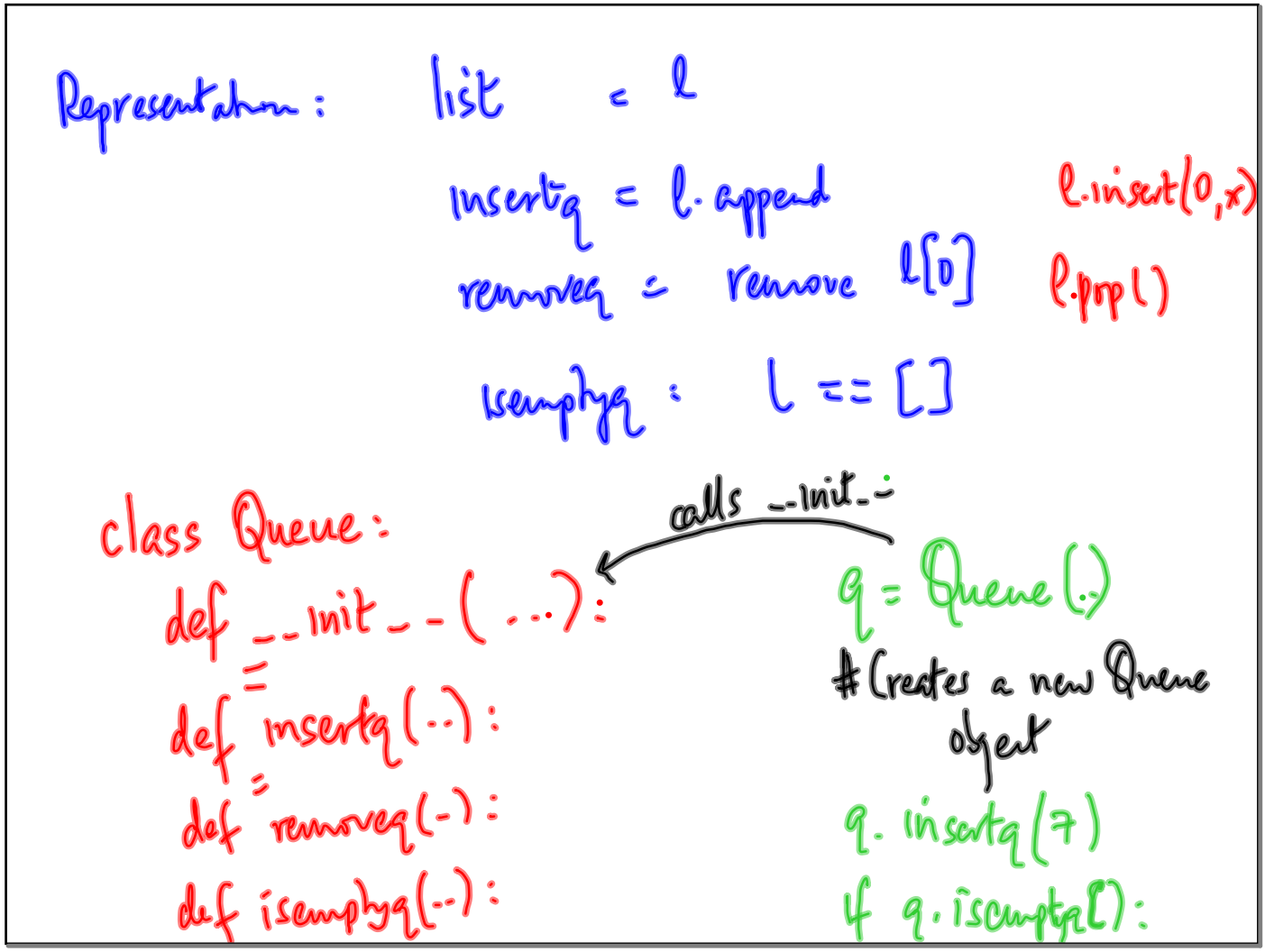
Create instances of the data structure and use public interface to manipulate them — Concrete values / OBJECT

Want to implement a queue



Interface:

- removeq() – return value at head, & remove
- insertq(x) – append x at tail
- isemptyq() – True iff queue is empty


Representation: list = l
insertq = l.append
l.insert(0,x)
removeq = remove l[0]
l.pop()
isemptyq: l == []
class Queue:
def __init__(...):
def insertq(..):
def removeq(..):
def isemptyq(..):
calls __init__
q = Queue(..)
# Creates a new Queue object
q.insertq(7)
if q.isemptyq():

sort(l) vs l.sort()

insertq(q, x) vs q.insertq(x)

Internals

Inside a class definition, each function operates implicitly wrt a given object

Referred to as "self"

Every function has an extra first argument that names the "self" object

```
class Queue:
    def __init__(self):
        self.l = []
    def insertq(self, x):
        self.l.append(x)
        return
```

```
def removeq(self):
    z = self.L[0]
    del(self.L[0])
    return(z)

def isemptyq(self):
    return(self.L == [])
```

Back to function definitions

```
def f(x,y,z):
    ...
```

Must call f with 3 args

```
f(a,b,c)
```

x=a, y=b, z=c

Can name the arguments

```
f(x=7, z=c, y=b)
```

Optional argument

```
x = int("17")          x → 17
```

optional

base

```
x = int("17", 8)       x →
```

```
def int(s, base=10):
```

default if no second argument is provided

Want to optionally create a Queue with some starting value

```
def __init__(self, initHist = []):
    self.l = initHist
```

```
q = Queue()
```

empty queue

```
q = Queue([3,4])
```

creates a queue with [3,4]