



Lectures in software design in python

Abel Carreras

Introduction

Why Programming (software) design?

Readability

Understandable
by others (and you)

Maintainability

Fixing bugs
Upgrade to new
environments

Extensibility/flexibility

Add new
features

Do not waste your time!

Introduction

Readability

- You may want to reuse your code in the future
- You may want to share your code
- You will to explain how to use your code to others
- You will not have much time to read your code
- You may not remember how your code works anymore

Make your code readable!

Introduction

Maintainability

- Your OS will become obsolete
- Your Python version will become obsolete
- Your computer will become obsolete
- You will not have time to update your code
- You may not remember how your code works anymore

Make your code intuitive!

Introduction

Extensibility

- You may want to add new features to your code
- These features may conflict with your current structure
- You will not have time to change your code structure
- You may not remember how your code works anymore

Make your code flexible!

Introduction

The truth

- Your code (design) is crap: no users (maybe you in recent future)
- Your code is bad: future you is your only user
- Your code is fine: future you is your main user
- Your code is good: future you among some other researchers are users
- Your code is great: many researchers in your field may be users (future you included)

This is also for you!

Introduction

What is a good design?

Like writing
a paper!

- **Logical and intuitive**
People do not like to read manuals (and probably neither do you)
- **Less comments and better code**
let the code speak for you
- **Divide and conquer**
Properly organize your code in files, modules, functions, paragraphs,...
- **Explicit better than implicit (but..)**
Simple and long better than short (compact) and complicated
- **(...) Make use of available good python modules**
do not reinvent the wheel. Better notation is always nice!

Logical and intuitive

How?

- **Use conventions to write: PEP8**
 - ***variable names:***
explicit names, lower case, spaces as underscore “_”
 - ***blank lines and spaces:***
use them to separate logical blocks in your code
- **Aim for a good equation-code correspondence**
write helper functions if necessary
- **Let the code guide the structure**
minimize module imports by playing with scopes

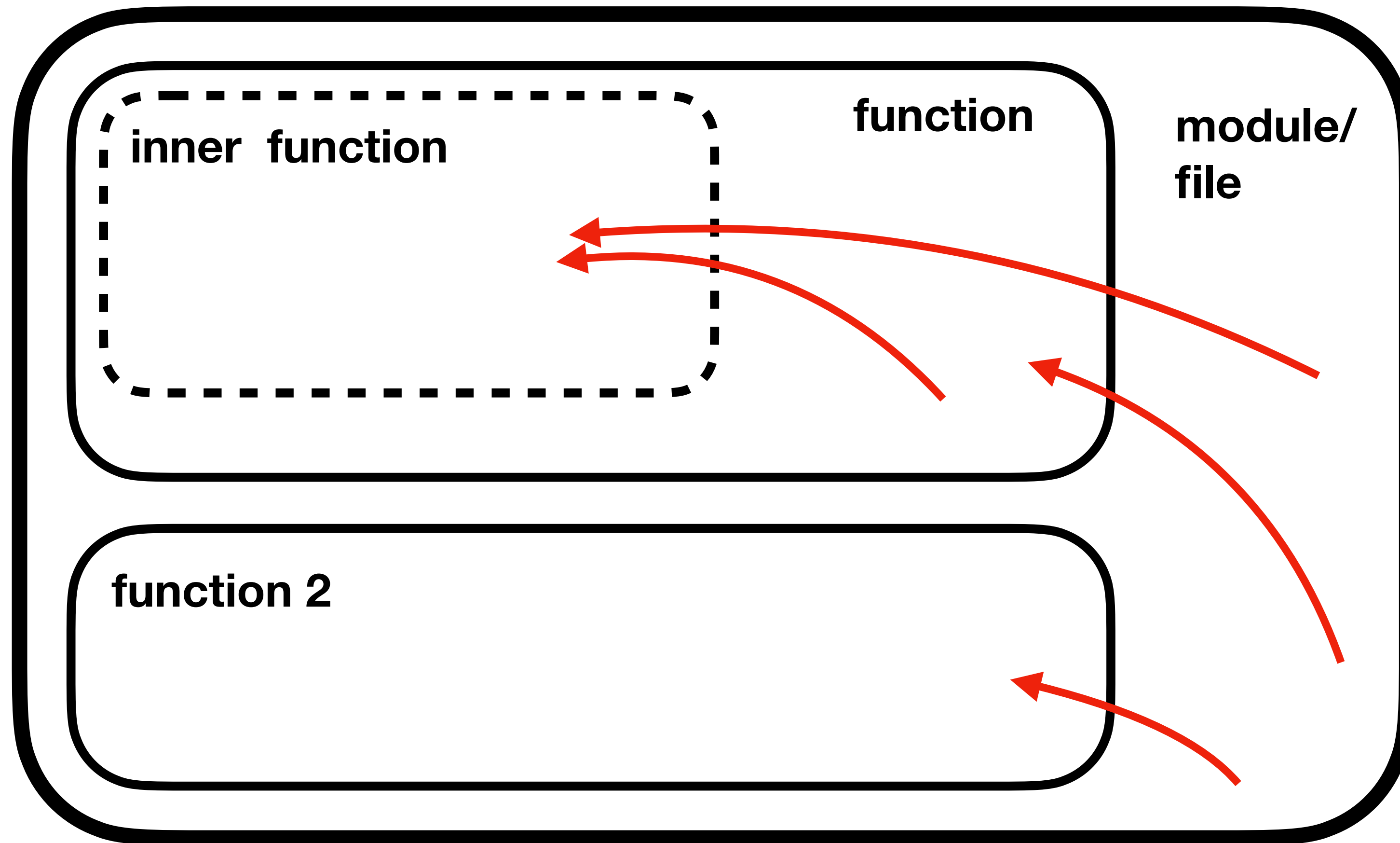
Example 1

Some initial advices

- **Use good code editor (IDE):**
PyCharm, Visual studio, Spider, etc..
- **Get used to a version control system (VCS) —> git**
very easy to use through IDE
- **Check the documentation (or StackOverflow)**
do your research before coding
- **Prioritize standard library**
over other obscure modules
- **Try to use widely compatible syntax**
support old versions of Python

Example 2

Variable Scope



Example 3

Generators/Iterators

generator \longrightarrow Iterator

```
def generator(n):  
    for i in range(n):  
        sum1 = 0  
        for j in range(5):  
            sum1 += i ** 2 + j  
  
        yield sum
```

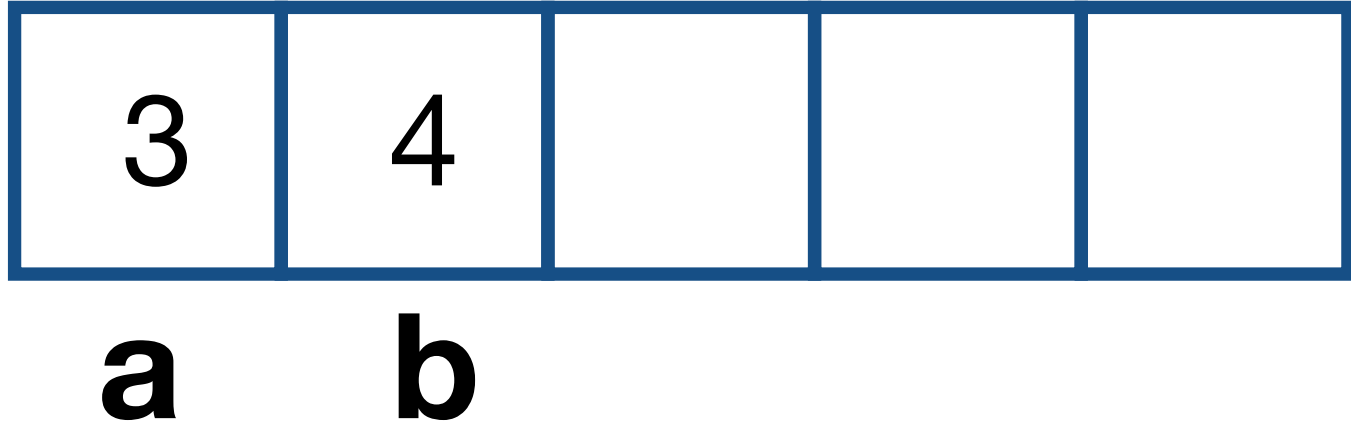
```
for element in generator(10):  
    print(element)
```

- iteration variables generated on the fly
- Memory efficient

Example 4

Fortran/C vs Python

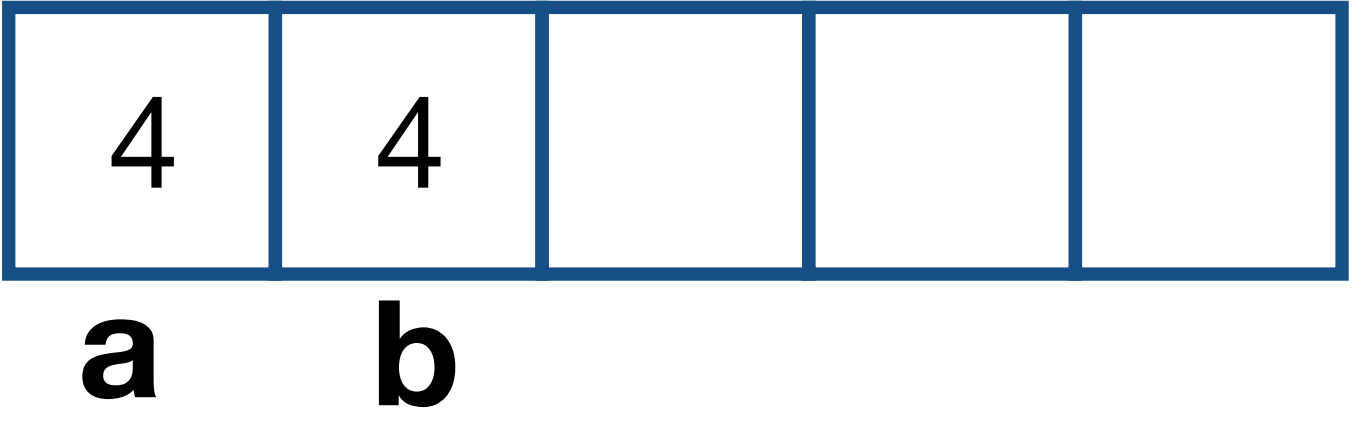
RAM Memory



```
a=3
b=4
```



RAM Memory



```
a=3
b=4
a=b
```


Fortran/C vs Python

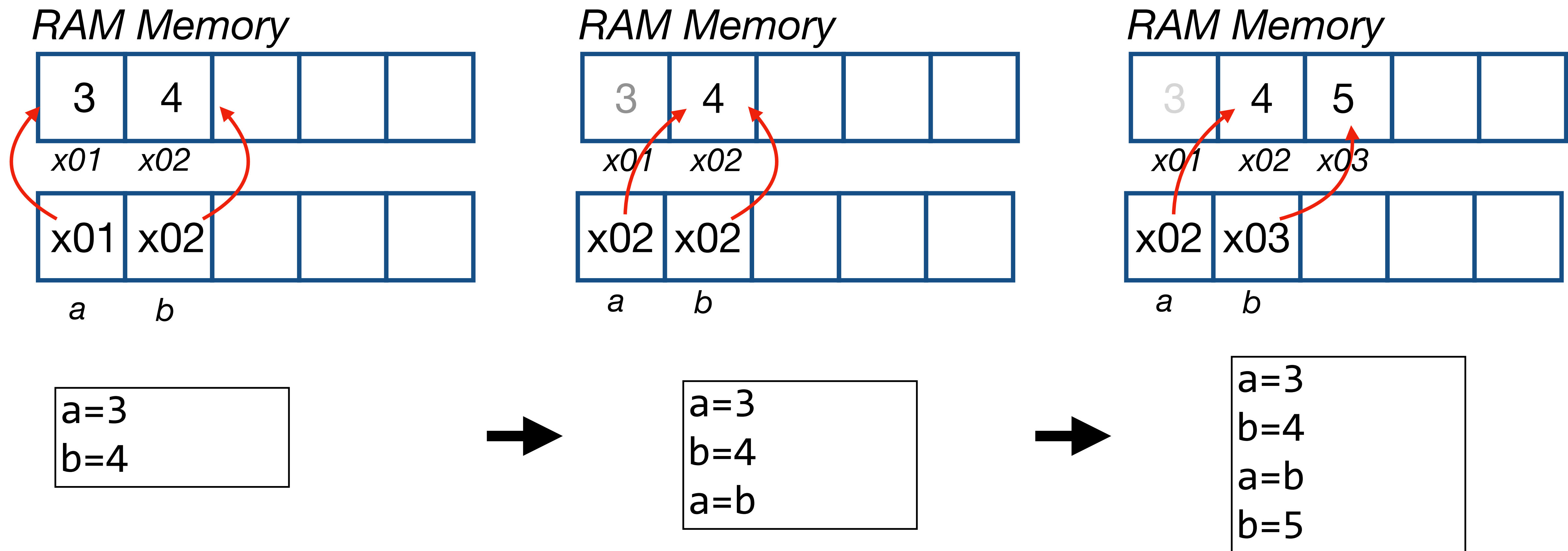
FORTRAN/C

- Small set of instructions
- Simple instructions very flexible
- Short documentation, lots of creativity
- Fast and insecure
- Compiled
- Not so nice syntax
- Fewer external libraries and not so easy to use

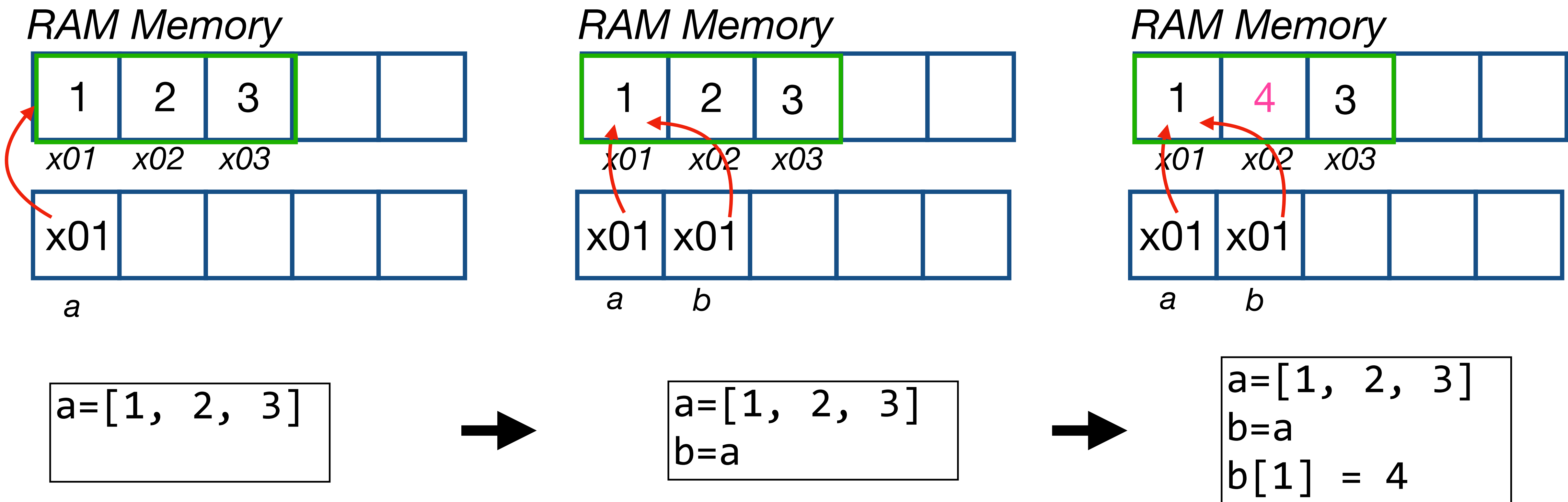
Python

- Large set of instructions
- Complex instructions and very specific
- Long documentation, lots of research (and creativity)
- Slow and safe
- Interpreted
- Nice syntax
- Lots of external libraries easy to use

Fortran/C vs Python



Fortran/C vs Python



Objects in Python

Object: Structured variable

- **Internal variables (internal state)**
Determine the properties of an object
- **Methods**
functions that act on the internal state
 - *Modifiers*
Modify the state of the object
 - *Informatives*
Return data based on the state of the object

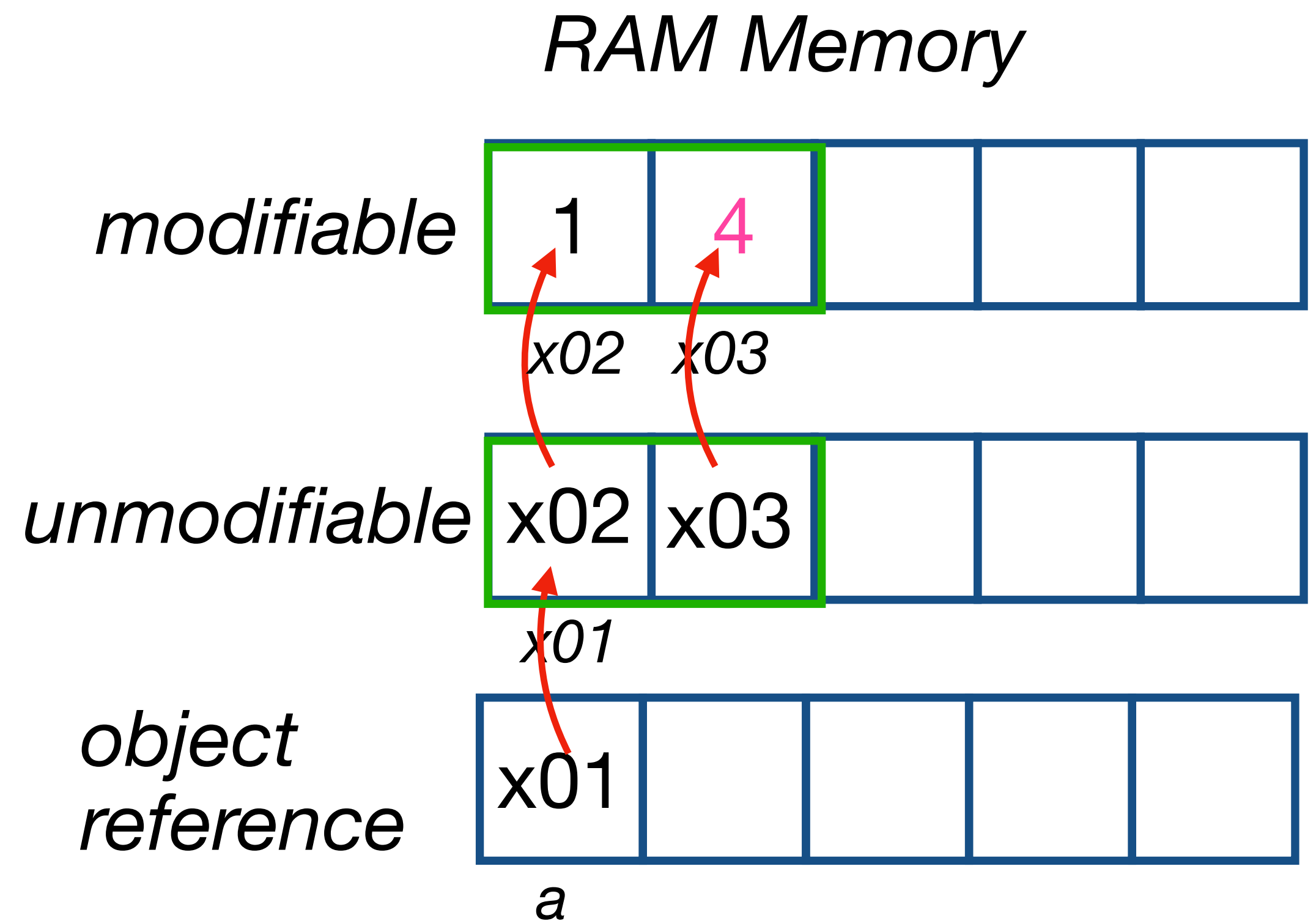
```
object.method()  
object.variable
```

Objects in Python

types of objects: *a security measure*

- **mutable**
can modify its contents
- **Un-mutable**
cannot modify its contents

Warning! an un-mutable object
can contain mutable contents



Objects in Python

Custom objects

Class

Contains the blueprints
to generate an object



Instance of a class

Object itself created
from a class

Example:

*The blueprints of a ship
(Blueprint type object)*

*The ship called “Titan”
(ship type object)*

Objects in Python

Getter/setter methods

- **getter**
get data from inner variables (get property)
- **setter**
set data to inner variables (modify object)

```
object.get_data()  
object.set_data(data)
```

Private/public methods

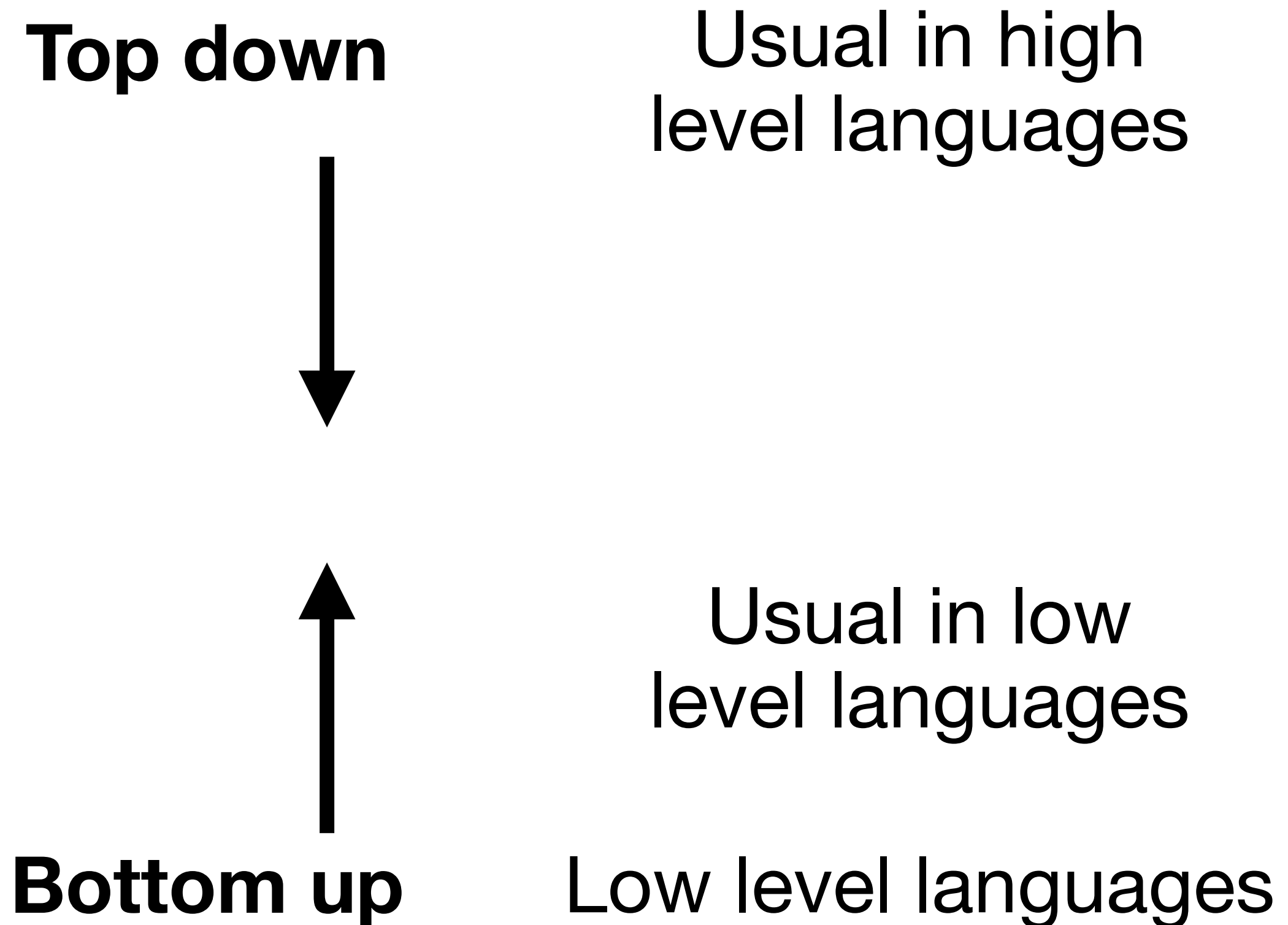
- **Private**
method used only in the same object (underscore)
- **Public**
general methods

```
object._get_method()  
object._set_method(data)
```

Example 5

Core design

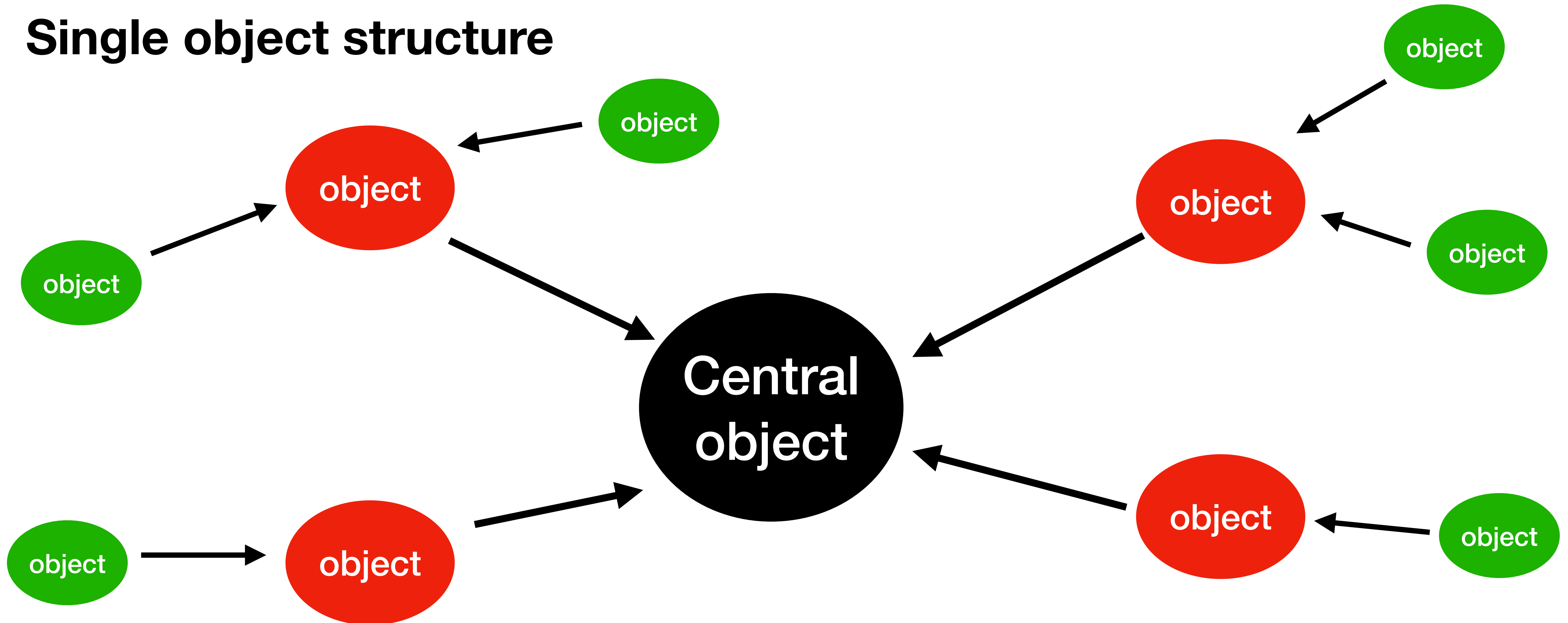
Programming strategies



Example 6

Core design

Single object structure



Core design

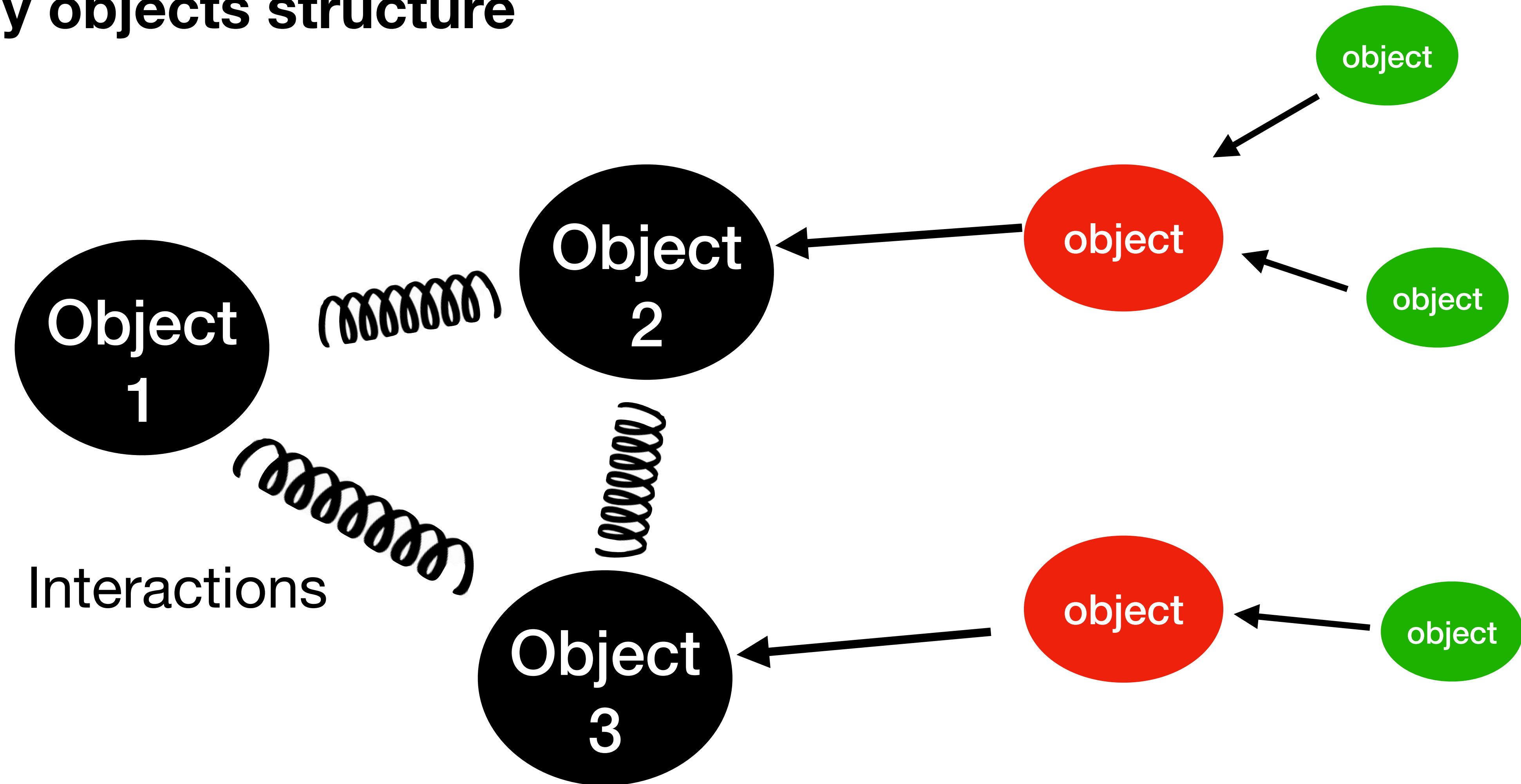
Single object structure

- Data collection (from file/terminal/..)
- Object creation (using data)
- Object modification
- Data extraction (from object)
- Data presentation

Example 7a

Core design

Many objects structure



Core design

Many objects structure

through **Functions**

Multiple objects
as arguments

```
data = interaction_function(object_1, object_2):  
print(data)
```

Multiple objects and other arguments

through **Operators**

Two objects
operate

```
data = object_1 * object_2  
print(data)
```

Only two objects, no arguments

Core design

Several interesting operators

- addition
- product
- length
- string representation
- get item
- iterator
- equal

```
self.__add__(self, other)  
self.__radd__(self, other)
```

```
self.__len__(self)  
self.__str__(self)  
self.__getitem__(self)
```

```
self.__iter__(self)  
self.__eq__(self, other)
```


Example 7b/8

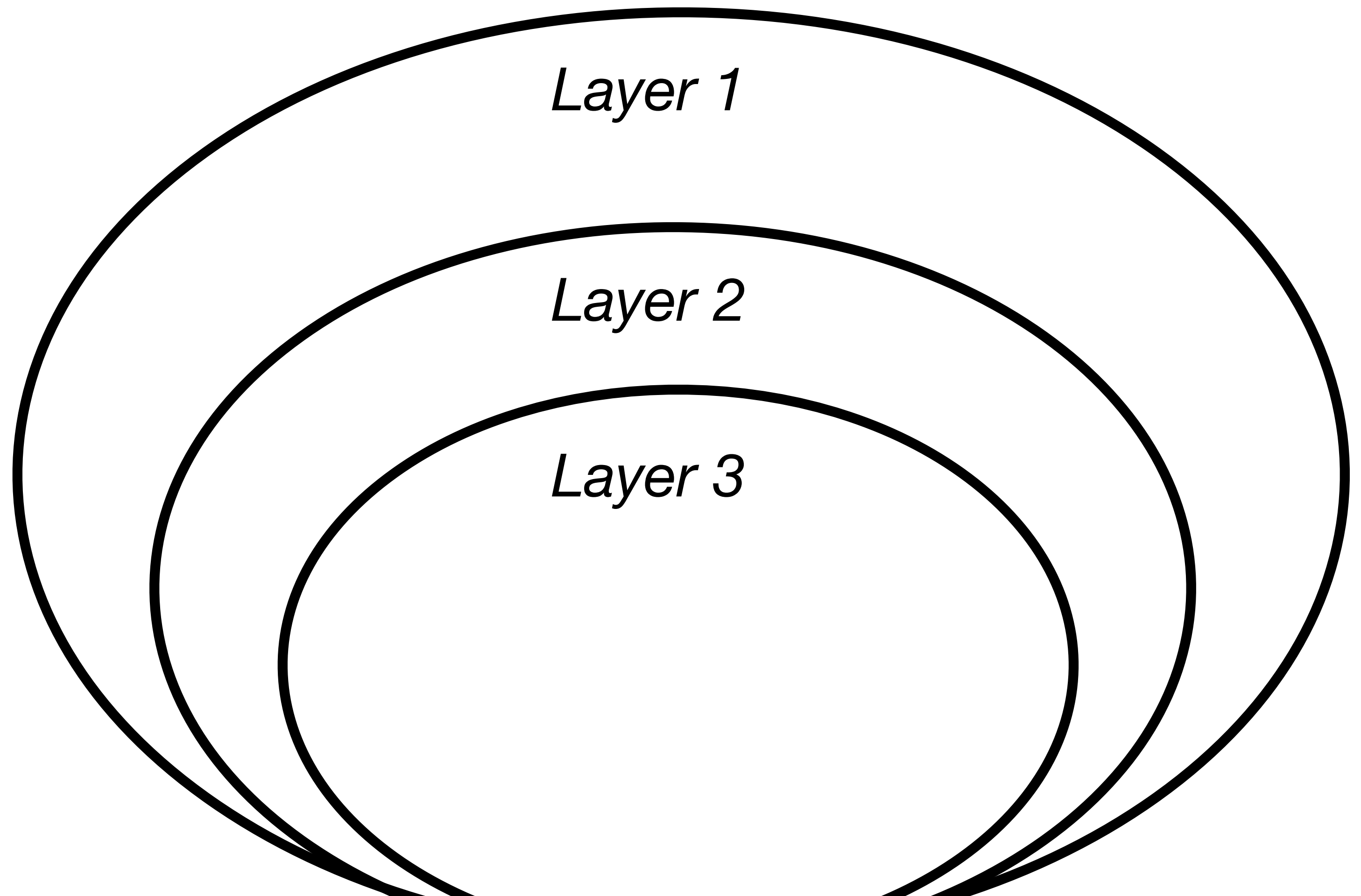
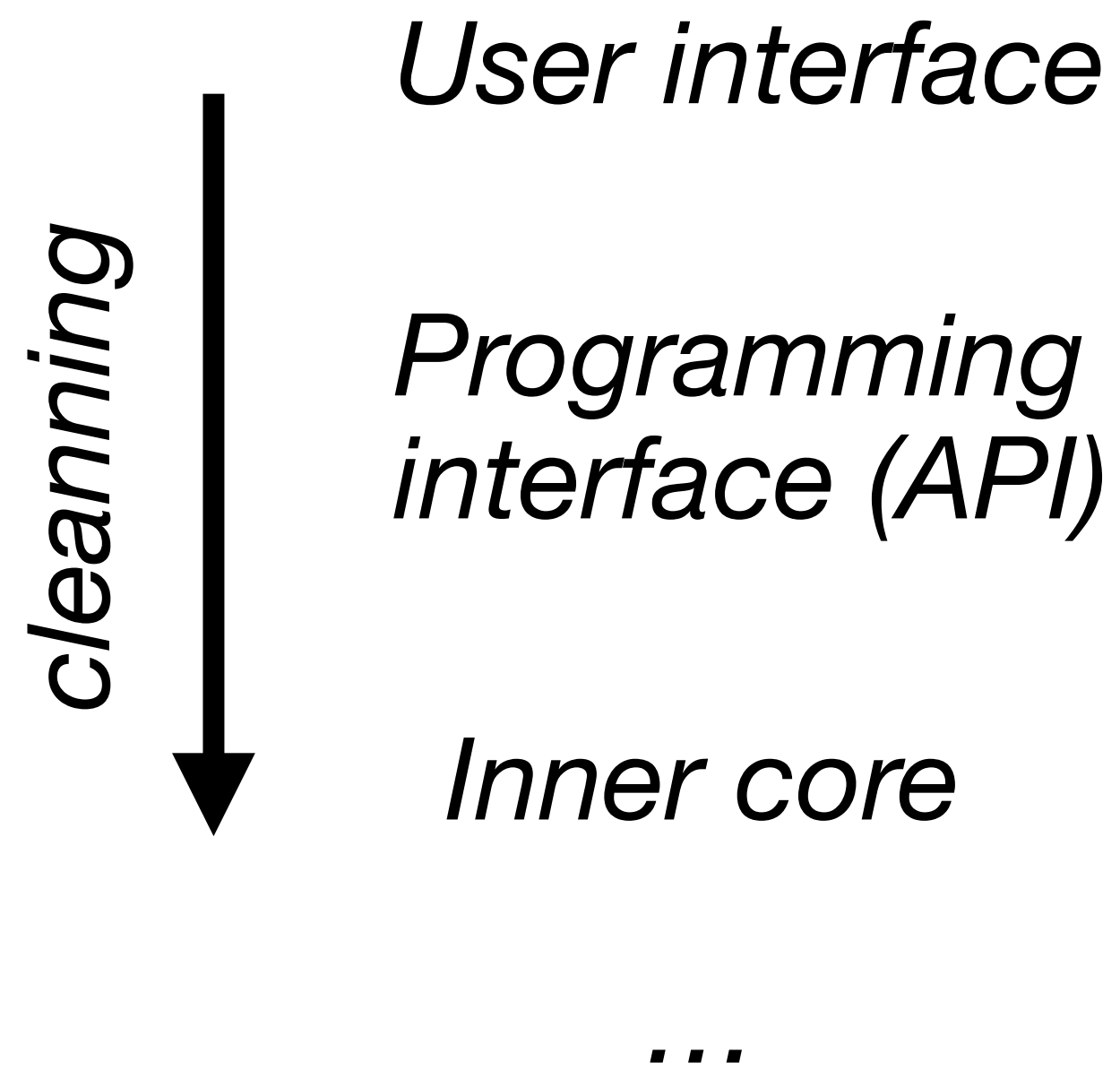
Divide and conquer

- **Polymorphism**
Objects mimic behavior of other objects
- **Abstraction**
Relation between real objects and Python objects
- **Encapsulation**
Separate the code in self contained parts

Example 7

Core upgrade

working in the code



Core upgrade

upgrade paths

List
simple
ordered
data



Dictionary
random access
non-ordered
data



Cluster class
complex
data structure