Building Data Structure

1

TOWARDS MORE ADVANCED CONCEPTS
&
OBJECT ORIENTED PROGRAMMING

Overview

- Data representation
- Building new data structures
- Examples

Data Representation

Example:

- Address
  - house number
  - street name
  - city
  - county
  - postcode
- Person
  - Surname
  - First name
  - NI
  - Address
  - Phone numbers

Using what we know so far

- Lists
  - Address

  [house_number, street_name, city, county, postcode]
  - Person

  [surname, first_name, NI, address, phone_numbers]

- String
  - Address

  “house_number, street_name, city, county, postcode”
  - Person

  “surname, first_name, NI, address, phone_numbers”

Using what we know so far

- Dictionary
  - Address

    {house_number:15, street_name : ‘Lili Street’,
    city: ‘York’, county: ‘Yorkshire’, postcode: ‘YO5-5GH’}

  - Person

    {surname: ‘Blot’, first_name: ‘Lilian’, NI: ‘OO7’, address:…,
    phone_numbers: {…} }

Creating our Own Data Structure

- Keyword `class`
- General Form:

Code

```
class DataStructureName:
      ``` doc-string ```
      pass
```

- Example

Code

```
class Address:
      ``` doc-string ```
      pass
```

Creating our Own Data Structure

- Using the new Data structure :

Code

```
>>> addr = Address()
>>> addr.street = 'Lili Street'
>>> print addr.street
Lili Street
```

What are Classes?

- **Classes** are composed from **structural** and **behavioural** constituents.
- **Attributes** (member variables or instance variables) enable a class instance to maintain state.
  - a.k.a *properties*, *fields*, *data members*
- **Methods**, enable the behaviour of class **instances**.
- **Classes** define the type of their instances

Creating our Own Data Structure

- Keyword `class`
- Key method `__init__(...)`
- Key parameter `self`

Code

```
class DataStructureName:
      ``` doc-string ```
      def __init__(self, <parameters>):
            <body>
```

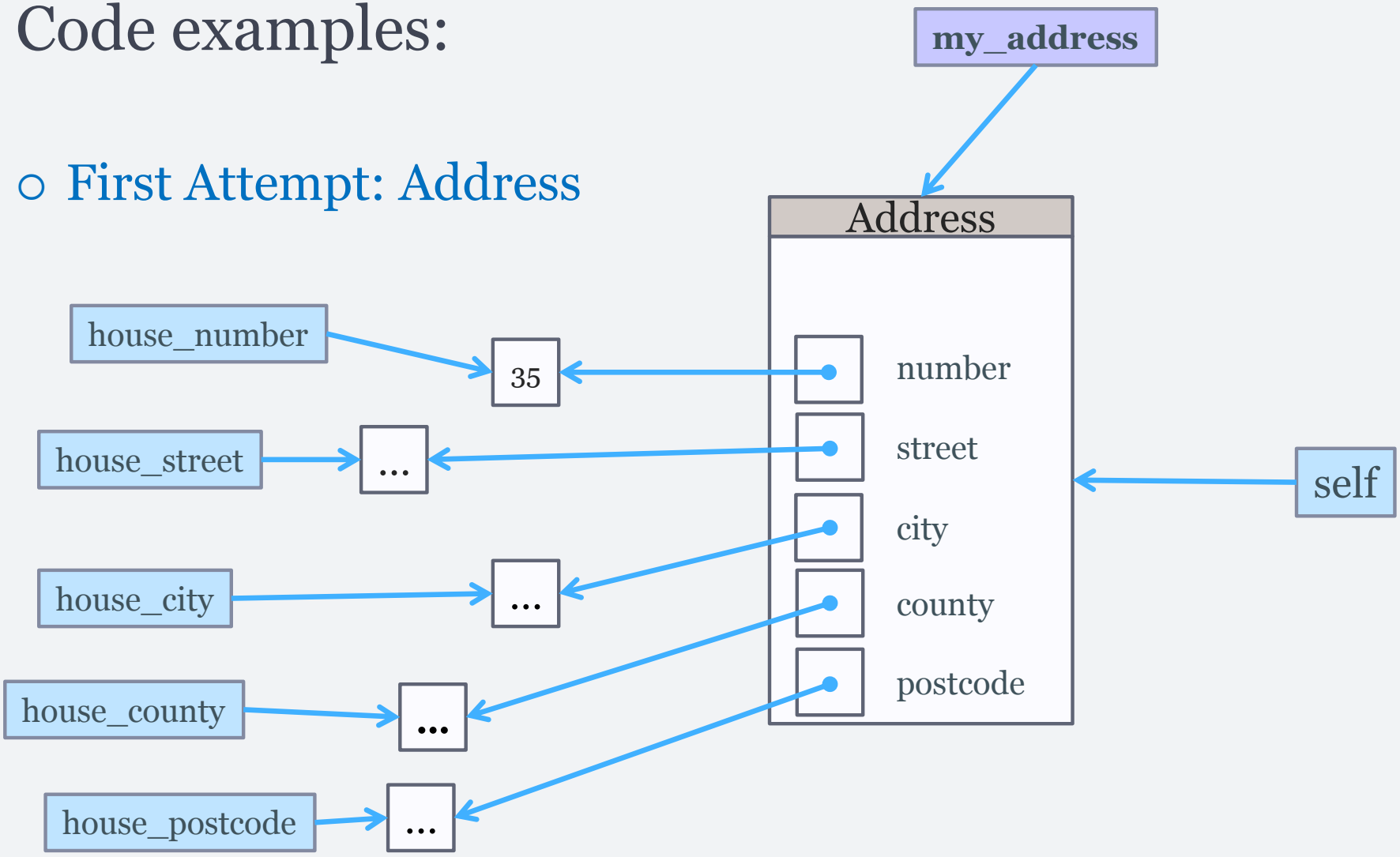
- `isinstance(object ,Class)`
```

Creating our Own Data Structure

- Code examples:
  - First Attempt: Address

Creating our Own Data Structure

- Code examples:
  - First Attempt: Address

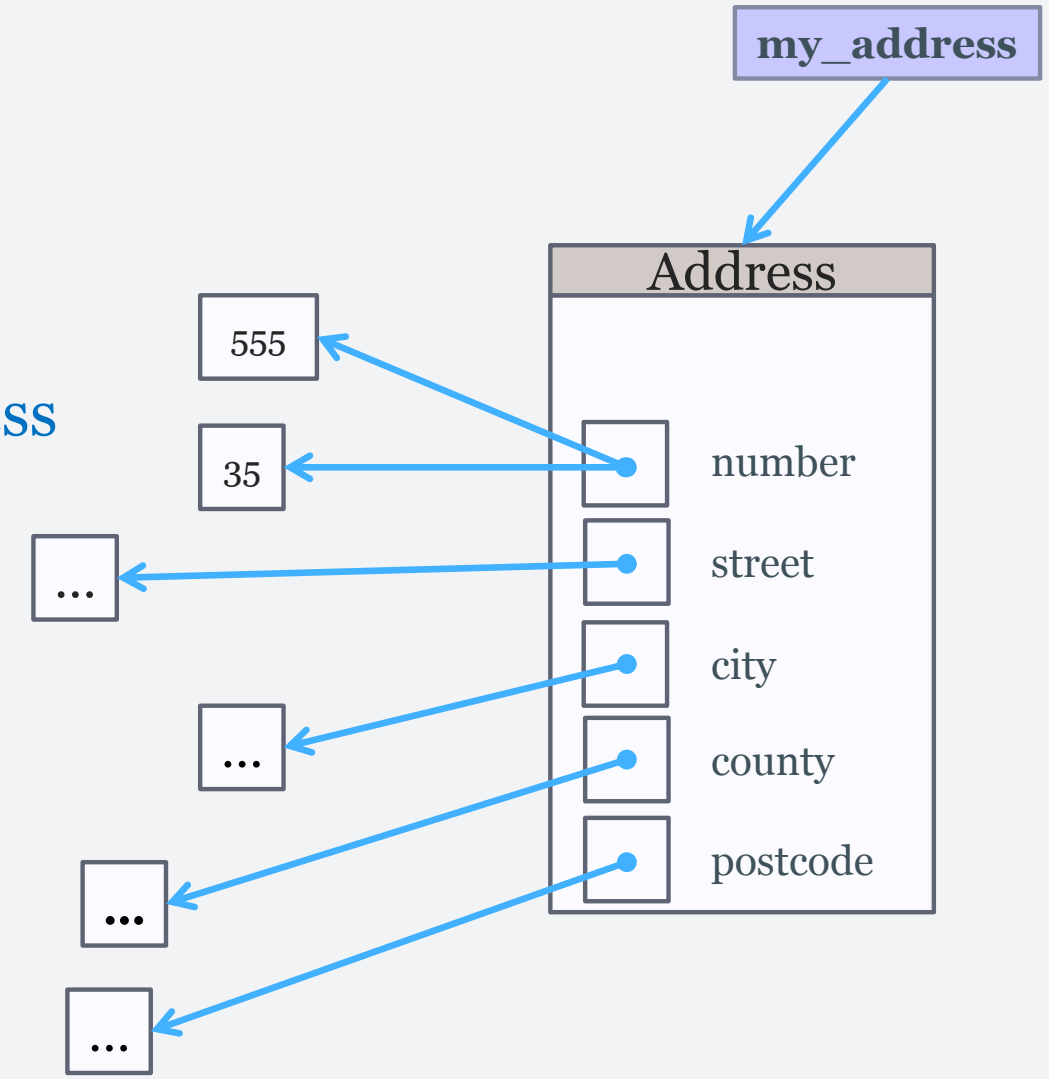


Creating our Own Data Structure

- Code examples:
  - First Attempt: Address
  - Second Attempt: Address

Creating our Own Data Structure

- Code examples:
  - First Attempt: Address
  - Second Attempt: Address



Creating our Own Data Structure

The objects we define are

MUTABLE

Terminology

- **Class** `Address` is the definition of an **object**
- `my_address` is **an instance of** `Address`
- `number`, `city`, `postcode`,... are **attributes**
- `__init__` is a **constructor**
- `__str__` is a **method**

Creating our Own Data Structure

- Code examples:
  - First Attempt: Address
  - Second Attempt: Address
  - Person

__repr__ Method

Definition:

Return a string containing a printable representation of an object. This is the same value yielded by conversions (reverse quotes). It is sometimes useful to be able to access this operation as an ordinary function. For many types, this function makes an attempt to return a string that would yield an object with the same value when passed to eval(), otherwise the representation is a string enclosed in angle brackets that contains the name of the type of the object together with additional information often including the name and address of the object. A class can control what this function returns for its instances by defining a __repr__() method.

___str___ Method

Definition :

Return a string containing a nicely printable representation of an object. For strings, this returns the string itself. The difference with repr(object) is that str(object) does not always attempt to return a string that is acceptable to eval(); its goal is to return a printable string. If no argument is given, returns the empty string, ''.

___str___ versus ___repr___

- Every class should implement the ___repr___ method
- ___str___ is optional. If not implemented, ___repr___ will be used instead
- ___repr___: representation of python object usually eval will convert it back to that object
- ___str___: is whatever you think is that object in text form

Summary

- We have seen how to build our own data structure
- Our data structure can be embedded into another data structure
- User defined Objects are MUTABLE

Exercises

Client requirements

- Build a genealogy tree
  - First Name, Surname, Maiden name
  - Date of Birth
  - Place of Birth (city/town/village and country)
  - Current Address

Your Job:

Build an appropriate data structure

Nested Structures

Exercises

Food for thought

- how to save/read the data structure into/from a text file
- how to search for/find a person in the tree
- how to add/remove a person from the tree
- could you implement your solution?