UNT

UNIVERSITY OF NORTH TEXAS®

Python for Data Science

Classes and Objects, Encapsulation, Abstraction

INFO 5502

Analytic Tools, Techniques and Methods

Ravi Varma Kumar Bevara

How and why would we define our own data types?

How and why would we define our own data types?

Classes!

Definition

class

A Python class defines a new data type for our programs to use.

Definition

class

A Python class defines a new **data type** for our programs to use.

*ints, strings, booleans, lists, floats, dictionaries, etc. are all **built-in** Python data types*

Definition

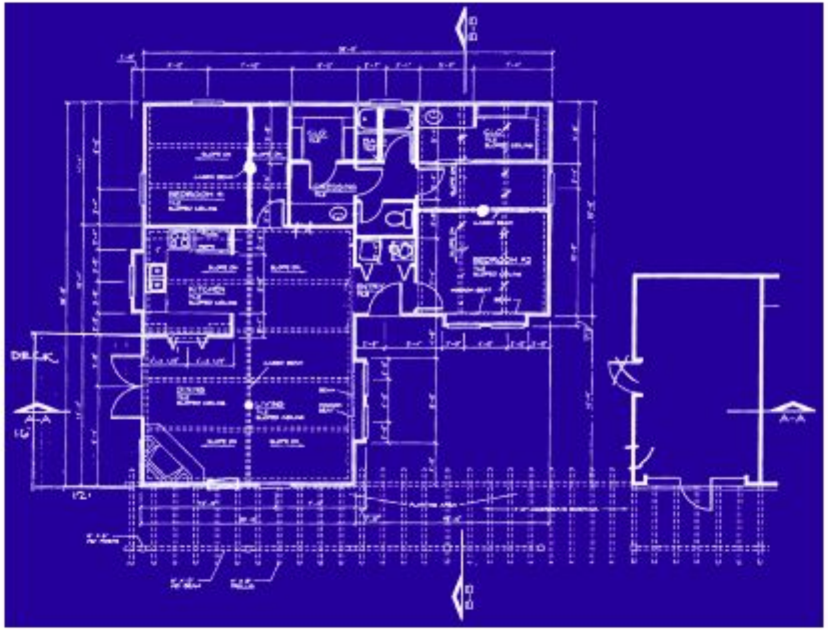
class

A Python class defines a new data type for our programs to use.

Classes allow us to define our own data types!

What is a class?

- A blueprint for a new type of Python **object**!



What is a class?

- A blueprint for a new type of Python **object**!
  - The blueprint describes a general structure, and we can create specific **instances** of our class using this structure.

What is a class?

- A blueprint for a new type of Python **object**!
  - The blueprint describes a general structure, and we can create specific **instances** of our class using this structure.

Definition

instance

When we create an object that is our new type, we call this creating an instance of our class.

What is a class?

- A blueprint for a new type of Python **object**!
  - The blueprint describes a general structure, and we can create specific **instances** of our class using this structure.

- 3 main parts
  - Attributes
  - Methods
  - Constructor

What is a class?

- A blueprint for a new type of Python **object**!
    - The blueprint describes a general structure, and we can create specific **instances** of our class.

- 3 main parts
    - Attributes (e.g. **`oval.fill_color`**, **`oval.width`**, etc.)
    - Methods
    - Constructor

Variables stored inside the class

What is a class?

- A blueprint for a new type of Python **object**!
  - The blueprint describes a general structure, and we can create specific **instances** of our class using this structure.

- 3 main parts
  - Attributes (e.g. **`oval.fill_color`**, **`oval.width`**, etc.)
  - Methods (e.g. **`oval.move()`**)
  - Constructor

Functions you can call on the object

What is a class?

- A blueprint for a new type of Python **object**!
  - The blueprint describes a general structure, and we can create specific **instances** of our class using this structure.

- 3 main parts
  - Attributes (e.g. **`oval.fill_color`**, **`oval.width`**, etc.)
  - Methods (e.g. **`oval.move()`**)
  - Constructor (e.g. **`GOval(width, height)`**)

How you create the object

How do we design a class?

We must specify the 3 parts:

How do we design a class?

We must specify the 3 parts:

1. Attributes: *What subvariables make up this new variable type?*

How do we design a class?

We must specify the 3 parts:

1. Attributes: *What subvariables make up this new variable type?*

Definition

instance attributes/instance variables

These variables belong to a specific instance of our class, and every new instance of our class can have its own values for each of them.

How do we design a class?

We must specify the 3 parts:

1. Attributes: *What subvariables make up this new variable type?*

```
instance.attribute
```

How do we design a class?

We must specify the 3 parts:

1. Attributes: *What subvariables make up this new variable type?*

```
image.width
```

How do we design a class?

We must specify the 3 parts:

2. Methods: *What functions can you call on a variable of this type?*

How do we design a class?

We must specify the 3 parts:

2. Methods: *What functions can you call on a variable of this type?*

Definition

methods

Methods are functions that belong to a class and can be called on objects that are of the type the class defines.

How do we design a class?

We must specify the 3 parts:

2. Methods: *What functions can you call on a variable of this type?*

```
instance.method(args)
```

How do we design a class?

We must specify the 3 parts:

2. Methods: *What functions can you call on a variable of this type?*

```
image.get_pixel(x, y)
```

How do we design a class?

We must specify the 3 parts:

3. Constructor: *What happens when you make a new instance of this type?*

Definition

constructor

A special kind of method that **instantiates** an object of your data type (i.e. creates an instance of your class)

How do we design a class?

We must specify the 3 parts:

3. Constructor: *What happens when you make a new instance of this type?*

```
instance = ClassName(args)
```

How do we design a class?

We must specify the 3 parts:

3. Constructor: *What happens when you make a new instance of this type?*

```
image = SimpleImage(width, height)
```

How do we design a class?

We must specify the 3 parts:

1. Attributes: *What subvariables make up this new variable type?*

2. Methods: *What functions can you call on a variable of this type?*

3. Constructor: *What happens when you make a new instance of this type?*

How do we design a class?

We must specify the 3 parts:

1. Attributes: *What subvariables make up this new variable type?*

2. Methods: *What functions can you call on a variable of this type?*

3. Constructor: *What happens when you make a new instance of this type?*

In general, classes are useful in helping us with complex programs where information can be grouped into objects.

Let’s create a social network for Python users!

`Pynstagram.py`

Let’s create a class to define a **`PynstaUser`**!

`PynstaUser`: We must specify our 3 parts

- Attributes?

`PynstaUser`: We must specify our 3 parts

- Attributes
    - Name (string)
    - Posts (list of strings)
    - Friends (list of other PynstaUsers)

`PynstaUser`: We must specify our 3 parts

- Attributes
  - Name (string)
  - Posts (list of strings)
  - Friends (list of other PynstaUsers)

- Methods?

`PynstaUser`: We must specify our 3 parts

- Attributes
  - Name (string)
  - Posts (list of strings)
  - Friends (list of other PynstaUsers)

- Methods
  - Post a status
  - Add a friend

`PynstaUser`: We must specify our 3 parts

- Attributes
  - Name (string)
  - Posts (list of strings)
  - Friends (list of other PynstaUsers)

- Methods
  - Post a status
  - Add a friend

- Constructor?

`PynstaUser`: We must specify our 3 parts

- Attributes
  - Name (string)
  - Posts (list of strings)
  - Friends (list of other PynstaUsers)

- Methods
  - Post a status
  - Add a friend

- Constructor: User should provide a username

`PynstaUser`: We must specify our 3 parts

- Attributes
  - Name (string)
  - Posts (list of strings)
  - Friends (list of other PynstaUsers)

- Methods
  - Post a status
  - Add a friend

- Constructor: `PynstaUser(name)`

Defining a class

```
class PynstaUser:
```

Defining a class

```
class PynstaUser:
```

Tells Python we're creating a new class

Defining a class

```
class PynstaUser:
```

The name of our new class

Style note

class names

Uppercase the first letter of every word in class names

Defining a class

```
class PynstaUser:

    def __init__(self):
```

The constructor method
for our class

Defining a class

```
class PynstaUser:

    def __init__(self, username):
```

Add the username as a parameter for our constructor

Defining a class

```
class PynstaUser:

    def __init__(self, username):
```

Whenever we create methods inside our class, we must pass in self

Defining a class

```
class PynstaUser:

    def __init__(self, username):
```

self refers to this specific instance of our class.

In other words, it makes sure that we're calling the method on the correct instance of our class!

Defining a class

```
class PynstaUser:

    def __init__(self, username):
        self.name = username
        self.friends = []
        self.posts = []
```

Create and initialize the attributes for *this instance* of the class

Defining a class

```
class PynstaUser:

    def __init__(self, username):
        self.name = username
        self.friends = []
        self.posts = []

    def add_friend(self, user):
        …

    def post(self, message):
        ...
```

We can define more methods for our class here!

Defining a class

```
class PynstaUser:

    def __init__(self, username):
        self.name = username
        self.friends = []
        self.posts = []

    def add_friend(self, user):
        …

    def post(self, message):
        ...
```

They must all take in self as a parameter

Summary

```
class PynstaUser:

    def __init__(self, username):
        self.name = username
        self.friends = []
        self.posts = []

    def add_friend(self, user):
        …

    def post(self, message):
        ...
```

Class definition and name

Constructor

Attributes (must start with self. to be attributes!)

Methods

How and **why** would we define our own data types?

Why do we use classes?

- For ourselves
  - Grouping related data and the functions that act on it
  - Modular code development (isolation of particular tasks)

- For others
  - We hide the implementation details of our code so others don’t need to worry about them.
  - They can just use the class, like we do for SimpleImage.

Why do we use classes in our own code?

Why do we use classes in our own code?

Encapsulation!

Definition

encapsulation

The process of grouping related information and relevant functions into one unit

Definition

encapsulation

The process of grouping related information and relevant functions into one **unit**

a class!

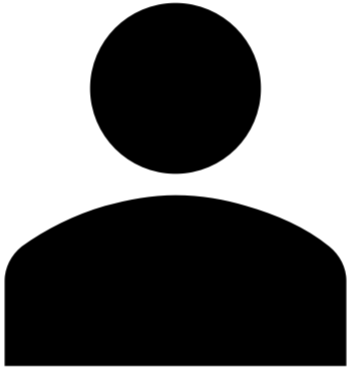
An analogy...

Suppose you’re a store owner looking to hire help to run your business. Here are some of the tasks that need to be covered:

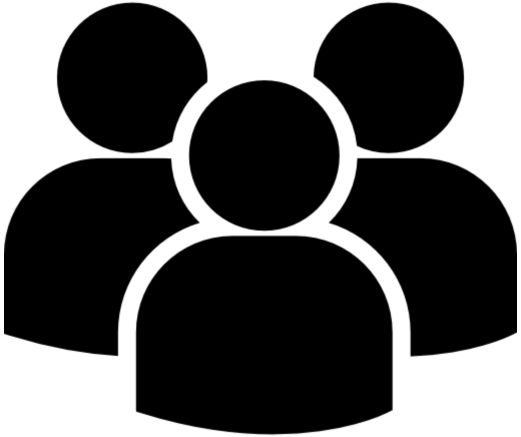
- Inventory management
- Cashier
- Advertising
- Customer service

An analogy…

Suppose you’re a store owner looking to hire help to run your business. You can either hire one person to do everything, or you can hire multiple people to split the work. Which do you choose?



vs.



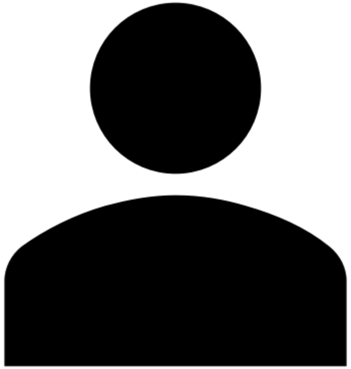
An analogy…

Suppose you're a store owner looking to hire help to run your business. You can either hire one person to do everything, or you can hire multiple people to split the work. Which do you choose?

- Easier for you to hire one person instead of multiple!
- Then you only need to deal with one person later on!

An analogy…

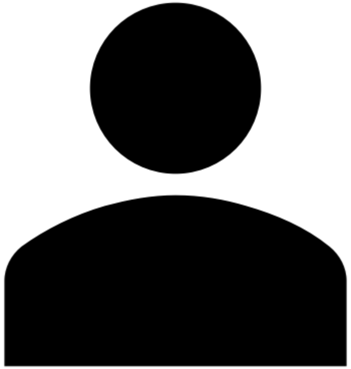
Suppose you're a store owner looking to hire help to run your business. You can either hire one person to do everything, or you can hire multiple people to split the work. Which do you choose?



- ~~Easier for you to hire one person instead of multiple!~~
- ~~Then you only need to deal with one person later on!~~

An analogy...

Suppose you’re a store owner looking to hire help to run your business. You can either hire one person to do everything, or you can hire multiple people to split the work. Which do you choose?



- If something goes wrong, it’s harder to know where the issue happened.

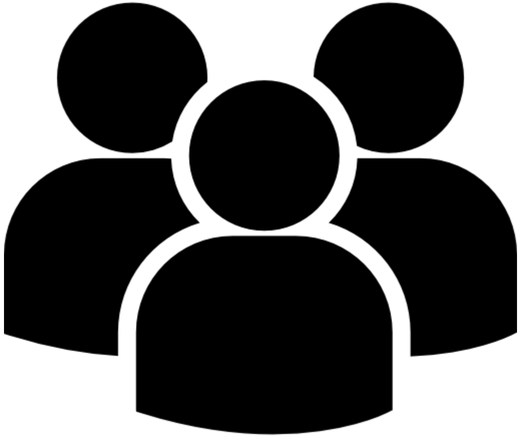
- You have to rely on that person for *everything*.

An analogy...

Suppose you're a store owner looking to hire help to run your business. You can either hire one person to do everything, or you can hire multiple people to split the work. Which do you choose?

- Splitting the work across people helps you know who's responsible for what.

- The individual people only need to know the information for their specific job.



Encapsulation via classes is like hiring multiple people!

Encapsulation via classes is like hiring multiple people!

- Integration
  - All the smaller parts add up to create the entire functionality
  - Similar to top-down decomposition

Encapsulation via classes is like hiring multiple people!

- Integration

- Modular development
  - You can separate different types of tasks and know where different information/functionality should be.
  - Easier for testing and debugging!

Encapsulation via classes is like hiring multiple people!

- Integration

- Modular development

- Instance variables (attributes)
  - Knowledge (data) for a specific class **stays inside** that class.
  - That information is **easier to access** across methods **within** that class.
  - If you need to access the information **outside** the class, there’s a **predefined structure** for doing so.

Encapsulation via classes is like hiring multiple people!

- Integration

- Modular development

- Instance variables (attributes)
  - Knowledge (data) for a specific class **stays inside** that class.
  - That information is **easier to access** across methods **within** that class.
  - If you need to access the information **outside** the class, there's a **predefined structure** for doing so.

Getters and setters!

Encapsulation enables **abstraction**!

- How classes help other people who use our code

- Classes as providing specific ways to interact with our code

Why do we use classes in code meant for others?

Why do we use classes in code meant for others?

Abstraction!

Definition

abstraction

Hiding implementation details of a class from the clients of that class

other programmers!

Clients and Interfaces

- Classes—or really any code we write (modules, libraries, etc.)—can be thought of from two perspectives.

Slide adapted from Jerry Cain

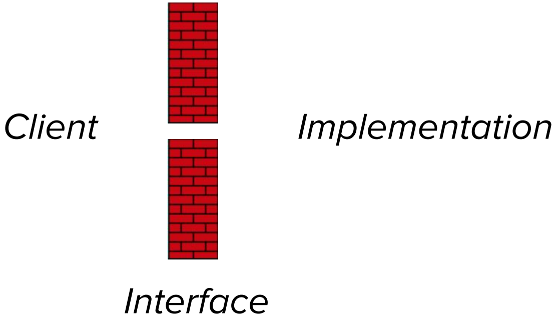
Clients and Interfaces

- Classes—or really any code we write (modules, libraries, etc.)—can be thought of from two perspectives.
- The point at which the client and implementation meet and communicate is known as the **interface**, which serves as both a barrier and a communication channel

Slide adapted from Jerry Cain

Clients and Interfaces

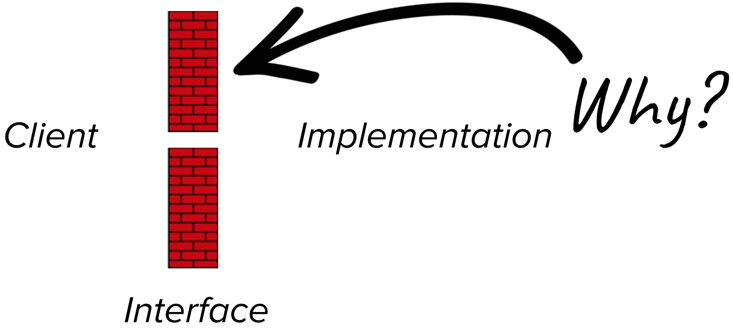
- Classes—or really any code we write (modules, libraries, etc.)—can be thought of from two perspectives.
- The point at which the client and implementation meet and communicate is known as the **interface**, which serves as both a barrier and a communication channel



Slide adapted from Jerry Cain

Clients and Interfaces

- Classes—or really any code we write (modules, libraries, etc.)—can be thought of from two perspectives.
- The point at which the client and implementation meet and communicate is known as the **interface**, which serves as both a barrier and a communication channel



Slide adapted from Jerry Cain

Information Hiding

- One of the central principles of modern software design is that each level of abstraction should hide as much complexity as possible from the layers that depend on it. This principle is called **information hiding**.

Slide adapted from Jerry Cain

Information Hiding

- One of the central principles of modern software design is that each level of abstraction should hide as much complexity as possible from the layers that depend on it. This principle is called **information hiding**.
- When you **use** a function, it is more important to know what the function does than to understand exactly how it works.

Slide adapted from Jerry Cain

Information Hiding

- One of the central principles of modern software design is that each level of abstraction should hide as much complexity as possible from the layers that depend on it. This principle is called **information hiding**.
- When you **use** a function, it is more important to know what the function does than to understand exactly how it works.
    - The underlying details are of interest only to the programmer who implements the function.

Slide adapted from Jerry Cain

Information Hiding

- One of the central principles of modern software design is that each level of abstraction should hide as much complexity as possible from the layers that depend on it. This principle is called **information hiding**.

- When you **use** a function, it is more important to know what the function does than to understand exactly how it works.
  - The underlying details are of interest only to the programmer who implements the function.
  - Clients who use that function as a tool can usually ignore the implementation altogether.

Slide adapted from Jerry Cain

Abstraction protects the data stored in an object

- Getters and setters are the interface to the data
  - These functions provide clients with a specific, limited way of accessing the data.
  - If clients could change the data in any way they wanted, things could get really messy.

Abstraction protects the data stored in an object

- Getters and setters are the interface to the data
  - These functions provide clients with a specific, limited way of accessing the data
  - If clients could change the data in any way they wanted, things could get really messy.

- Clients don’t have to worry about constraints on the data
  - The implementation will handle that for them behind-the-scenes!
  - E.g. A **`PynstaUser`** shouldn’t be able to add a friend they’re already friends with.

Abstraction protects the data stored in an object

- Getters and setters are the interface to the data
- Clients don’t have to worry about constraints on the data

An example!

PyPal.py

[abstraction demo]