

## objects, classes, and methods.

Week 10 | Lecture 1 (10.1)

if nothing else, write `#cleancode`

# This Week's Content

- **Lecture 10.1**
  - objects, classes, and methods
  - Reading: Chapter 14
- **Lecture 10.2**
  - Classes in Classes, Functions, and Collections
  - Reading: Chapter 14
- **Lecture 10.3**
  - Design Problem: Stock Market

# Procedural Programming

## Global Variable

Pedestrian 1  
x, y Location

## Global Variable

Pedestrian 2  
x, y Location

## Global Variable

Pedestrian 3  
x, y Location

```
x_ped1 = 3  
y_ped1 = 5
```

## Global Variable

Traffic Light 1  
Color

## Global Variable

Car 1  
x, y location

```
traffic_light = 'red'
```

## Separation of Data and Functions

## Function

OK to Cross

## Function

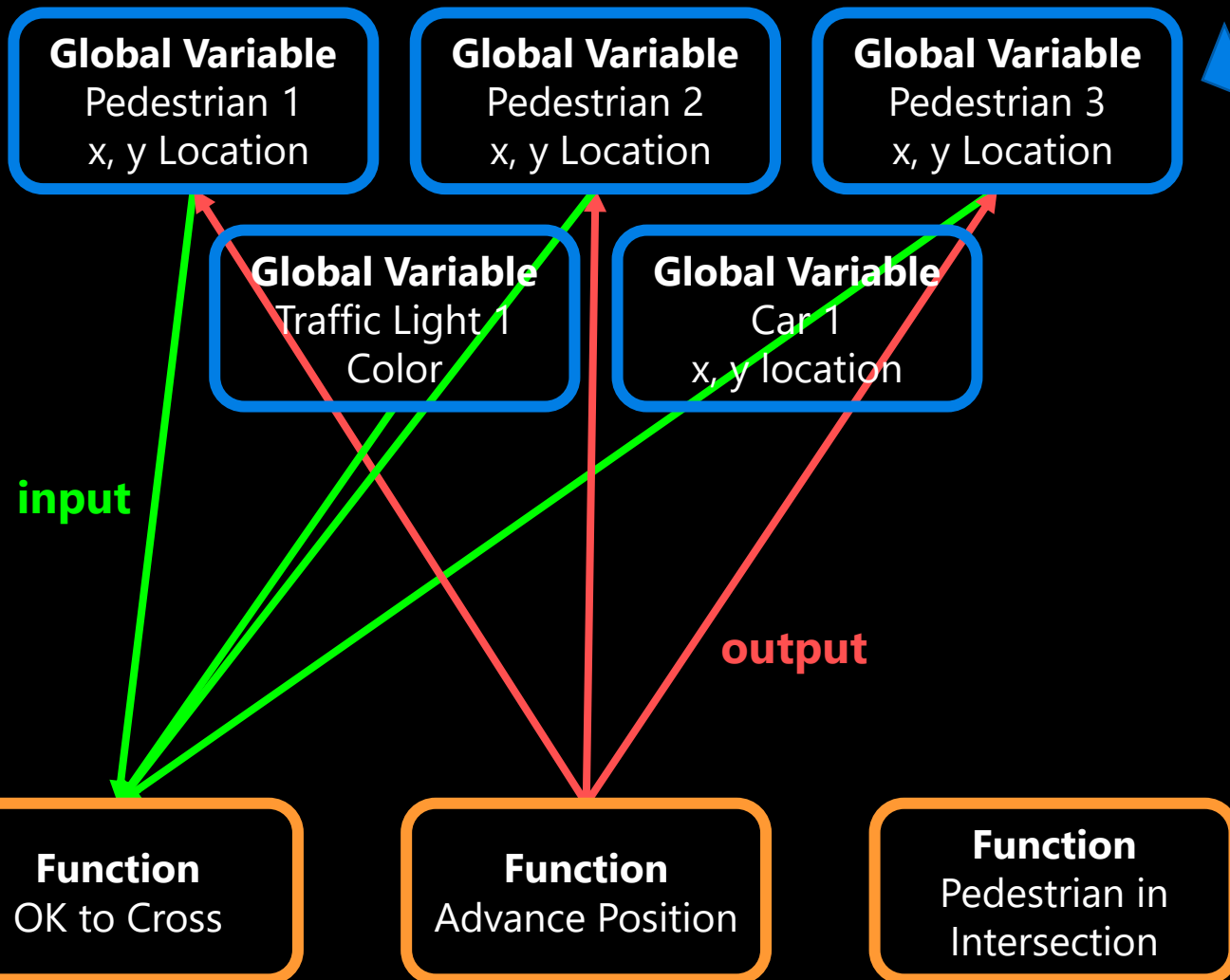
Advance Position

## Function

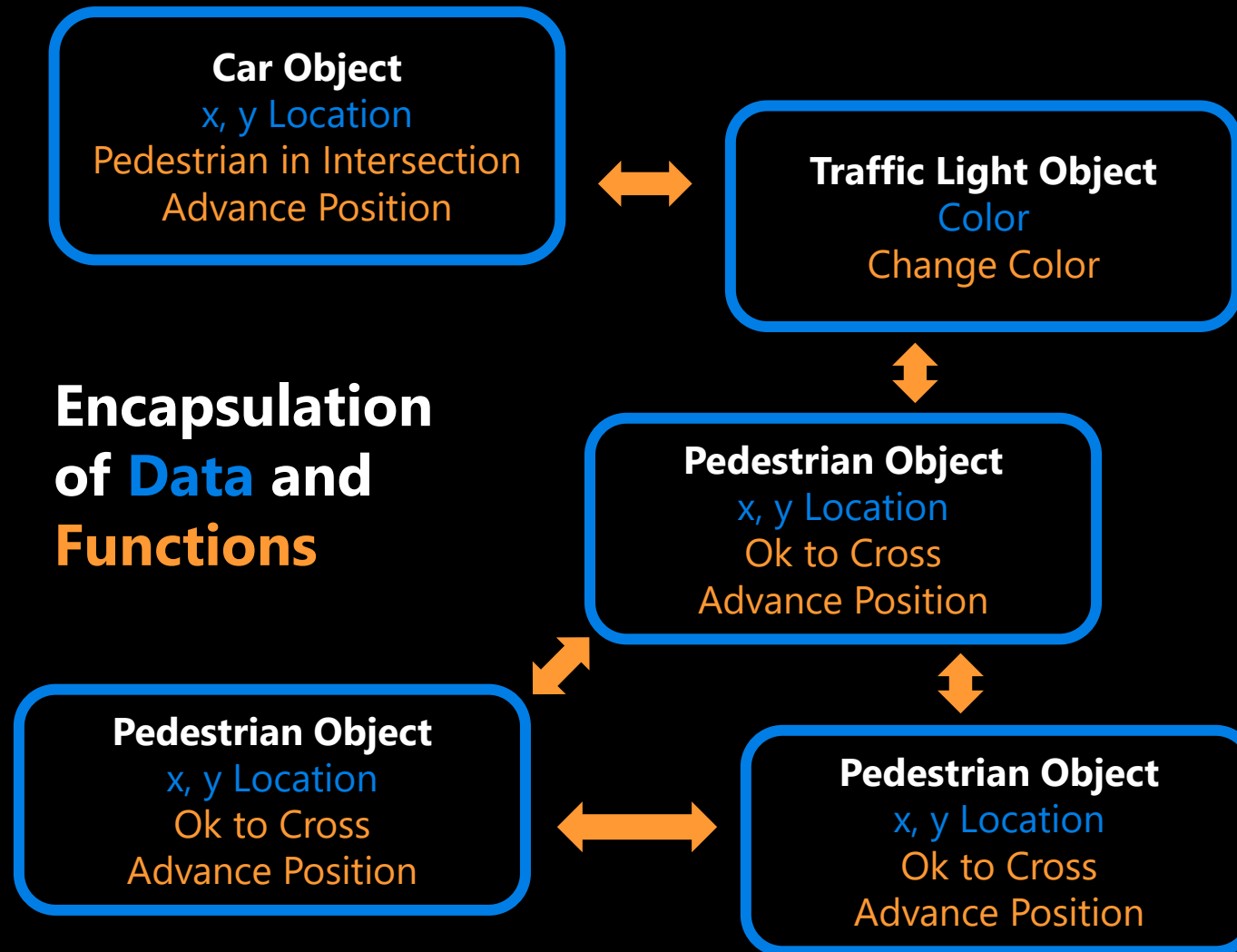
Pedestrian in  
Intersection



# Procedural Programming



# Object-Oriented Programming



# Object-Oriented Programming

- Often, an object definition corresponds to some object or concept in the real world.
- The functions that operate on that object correspond to the ways real-world objects interact.
- Examples:
  - **Oven Object:** the oven allows several specific operations, e.g., set the temperature, set a timer, etc.
  - **Cellphone Object:** we use a cellphone's own "methods" to send a text message, or to change its state to silent.
  - **Turtle Object:** we use a turtle's own "methods" to move around a 2D space.

# Object-Oriented Programming

## Data Functions

### Procedural

```
def up(y):  
    return y + 1  
  
def goto(x_new, y_new):  
    return x_new, y_new  
  
def right(x):  
    return x + 1
```

```
x = 0  
y = 0  
  
y = up(y)  
x, y = goto(-150, 100)  
x = right(x)  
  
print(x, y)
```

### Object-Oriented

```
alex = Turtle(0, 0)  
  
alex.up()  
alex.goto(-150, 100)  
alex.right()  
  
print(alex.x, alex.y)
```



# Objects in Python

- **Everything in Python is an object.**
- Every value, variable, function, etc., is an object.
- Every time we create a variable we are making a new object.

Is **this** an instance  
of **this** class.



```
>>> isinstance(4, object)
True
```

```
>>> isinstance(max, object)
True
```

```
>>> isinstance("Hello", object)
True
```



# Objects in Python

- Each object has a type or **class** it is associated with.

Is **this** an instance  
of **this** class.



```
>>> isinstance("Hello", str)
```

```
True
```

```
>>> isinstance(4, int)
```

```
True
```

```
>>> isinstance(4, float)
```

```
False
```

```
>>> isinstance(4.0, float)
```

```
True
```

```
>>> isinstance([1, 2], list)
```

```
True
```

# Objects in Python

- Really? Everything is an object?

**Open your  
notebook**

**Click Link:**

**1. Objects in Python**

# Classes

- A class can be thought of as a template for the objects that are instances of it.
- An instance of a class refers to an object whose type is defined as the class.
- The words "instance" and "object" are used interchangeably.
- A **Class** is made up of attributes (**data**) and methods (**functions**).

**Data**



**Attributes**

**Functions**



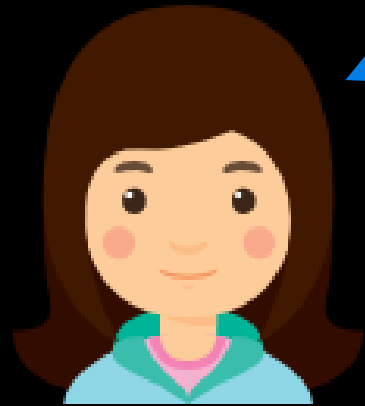
**Methods**

```
append(list1, list2)
```

```
list1.append(list2)
```



# Classes

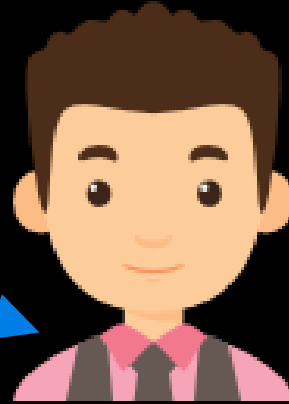


name: June  
age: 34  
city: Ottawa  
gender: she/her

Instances  
(objects) of the  
Person class.



name: Ted  
age: 31  
city: Kingston  
gender: he/him



name: Majid  
age: 28  
city: Toronto  
gender: they/them

Person

name  
age  
city  
gender

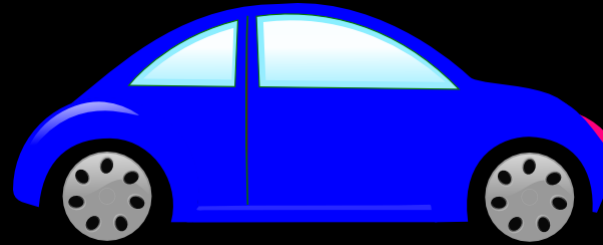
eat  
study  
sleep  
play

# Classes



**model:** Corolla  
**company:** Toyota  
**year:** 1980  
**color:** red

Instances  
(objects) of the  
Car class.



**model:** Model S  
**company:** Tesla  
**year:** 2017  
**color:** blue



**model:** Bus  
**company:** Volkswagen  
**year:** 1976  
**color:** orange

**Car**

**model**  
**company**  
**year**  
**color**

**brake**  
**accelerate**  
**change oil**  
**open trunk**

# Classes

Instances  
(objects) of the  
**Turtle** class.



**name:** Lucy  
**x location:** 24  
**y location:** 35



**name:** Susmit  
**x location:** 134  
**y location:** 45



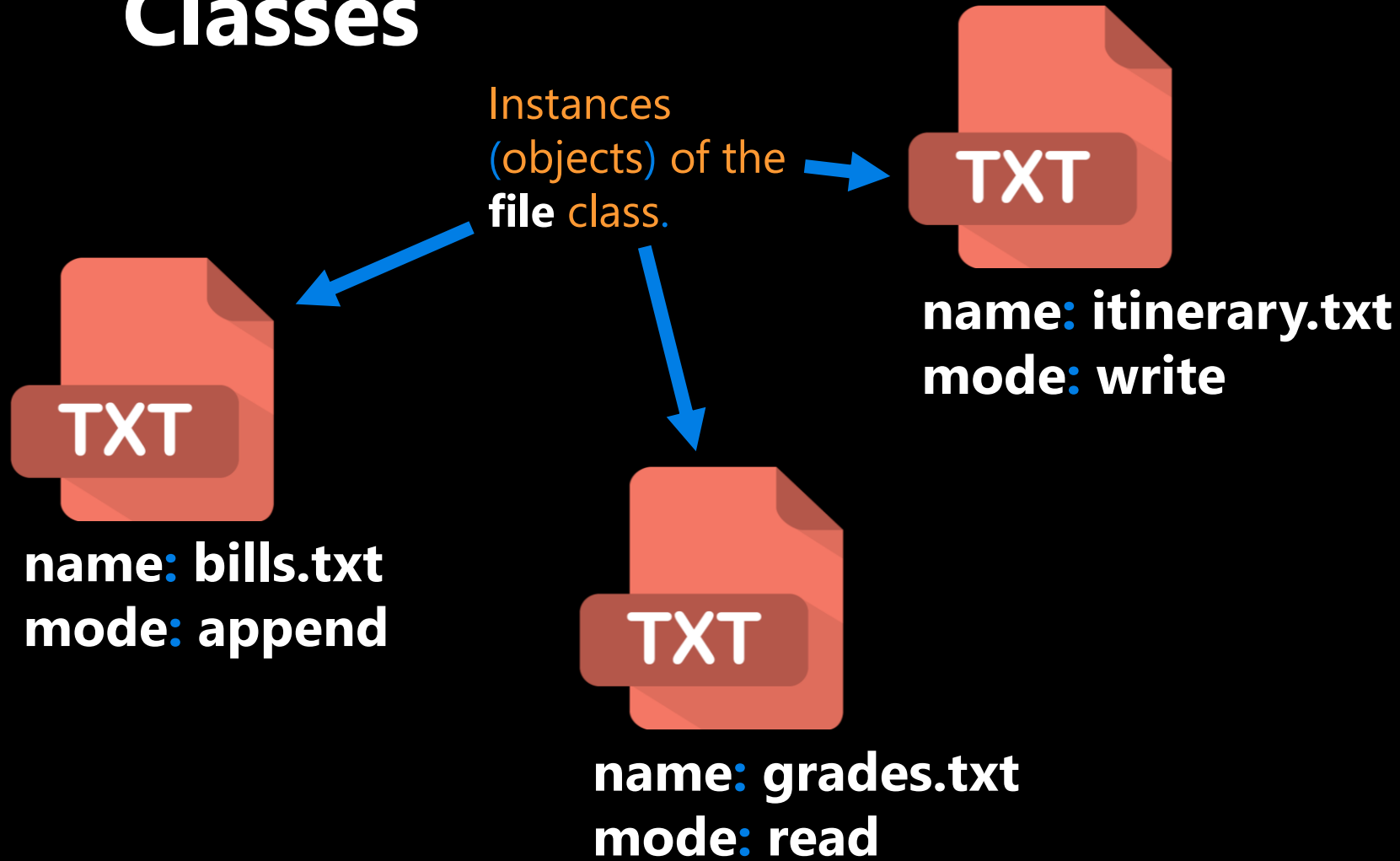
**name:** Brian  
**x location:** 92  
**y location:** 62

**Turtle**

**name**  
**x location**  
**y location**

**move up**  
**move down**  
**move left**  
**move right**  
**go to**

# Classes



**File**

**name**  
**mode**

**read**  
**readline**  
**readlines**



# Classes

- General form of a Class:

- Class Name
  - CamelCase
  - CourseGrades
  - BankAccount
  - FlightStatus
  - XRayImage
- Constructor
- Methods

```
class Name:
```

```
    def __init__(self, param1, param2, ...):  
        self.param1 = param1  
        self.param2 = param2
```

```
    ...  
    body
```

```
    def method1(self, parameters):  
        body
```

```
    def method2(self, parameters):  
        body
```

```
    def method3(self, parameters):  
        body
```

# Classes



```
alex = Turtle(0, 0)
```

```
alex.up()
```

```
alex.goto(-150, 100)
```

```
alex.down()
```

```
print(alex.x, alex.y)
```

```
class Turtle:
```

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

```
    def up(self):  
        body
```

```
    def goto(self, x, y):  
        body
```

```
    def down(self):  
        body
```

# Definition Recap

- Let's formally cover some important definitions.

**Class**

**Object**

**Instantiate**

**Method**

**Attribute**

**Constructor**

**self**

# Definition Recap

- Template for creating objects.

```
class Name:

    def __init__(self, param1, param2, ...):
        self.param1 = param1
        self.param2 = param2
        ...
        body

    def method1(self, parameters):
        body

    def method2(self, parameters):
        body

    def method3(self, parameters):
        body
```

**Class**

Object

Instantiate

Method

Attribute

Constructor

self

# Definitions

- An instance of a class.



```
alex = Turtle(0, 0)
```

```
class Turtle:
```

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

```
    def up(self):  
        body
```

```
    def goto(self, x, y):  
        body
```

```
    def down(self):  
        body
```

 **alex** is an instance of the **Turtle** class.

Class

Object

Instantiate

Method

Attribute

Constructor

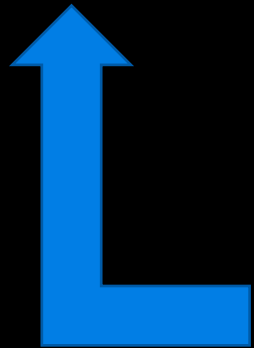
self

# Definition Recap

- Creating (constructing) an instance of a class.



```
alex = Turtle(0, 0)
```



This is the process of instantiating.

Class

Object

**Instantiate**

Method

Attribute

Constructor

self

# Definition Recap

`print()` is a function  
`list.append()` is a method

- A function defined in a class.

```
def goto(x, y):  
    body
```



Function

```
class Turtle:
```

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

```
    def goto(self, x, y):  
        body
```



Method

Class

Object

Instantiate

Method

Attribute

Constructor

self



# Definitions

- A variable bound to an instance of a class.



```
alex = Turtle(0, 0)
```

```
alex.x
```

```
alex.y
```

```
class Turtle:
```

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

```
    def up(self):  
        body
```

```
    def goto(self, x, y):  
        body
```

```
    def down():  
        body
```

Attributes

Class

Object

Instantiate

Method

Attribute

Constructor

self

# Definitions

- Responsible for setting up the initial state of a new instance.



```
alex = Turtle(0, 0)
alex.x
alex.y
```

```
class Turtle:
```

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

```
    def up(self):
        body
```

```
    def goto(self, x, y):
        body
```

```
    def down(self):
        body
```

`__init__` method is automatically run during instantiation.

Class

Object

Instantiate

Method

Attribute

Constructor

self

# Definitions

- Reference to the instance of the class.
- Although you do not technically need to use the word **self**, it is widely adopted and is recommended.
- Understanding **self** is a challenge for most students so don't worry if you're confused.
- More on **self** in the next lecture.

```
class Turtle:
```

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

```
    def up(self):  
        body
```

```
    def goto(self, x, y):  
        body
```

```
    def down(self):  
        body
```

Class

Object

Instantiate

Method

Attribute

Constructor

**self**

# Definitions

`katia is self`

**Inside Class**  
`self.attribute`  
`self.method`

**Outside Class**  
`katia.attribute`  
`katia.method`

```
katia = Turtle(0, 0)
```

```
katia.up()
```

```
class Turtle:
```

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

```
    def up(self):  
        ...
```

```
    def goto(self, x, y):  
        ...  
        ...
```

```
    def get_position(self):  
        ...
```

```
    def print_position(self):  
        ...
```

Class

Object

Instantiate

Method

Attribute

Constructor

**self**

# Definitions

`katia is self`

**Inside Class**  
`self.attribute`  
`self.method`

**Outside Class**

`katia.attribute`  
`katia.method`

`katia = Turtle(0, 0)`

`katia.up()`

```
class Turtle:
```

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

```
    def up(self):  
        self.y += 1
```

```
    def goto(self, x, y):  
        ...  
        ...
```

```
    def get_position(self):  
        ...
```

```
    def print_position(self):  
        ...
```

Class

Object

Instantiate

Method

Attribute

Constructor

**self**

# Definitions

`katia is self`

**Inside Class**  
`self.attribute`  
`self.method`

**Outside Class**

`katia.attribute`  
`katia.method`

```
katia = Turtle(0, 0)
```

```
katia.up()  
katia.goto(-2, 10)
```

```
class Turtle:
```

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

```
    def up(self):  
        self.y += 1
```

```
    def goto(self, x, y):  
        ...  
        ...
```

```
    def get_position(self):  
        ...
```

```
    def print_position(self):  
        ...
```

Class

Object

Instantiate

Method

Attribute

Constructor

**self**

# Definitions

`katia is self`

**Inside Class**  
`self.attribute`  
`self.method`

## Outside Class

`katia.attribute`  
`katia.method`

```
katia = Turtle(0, 0)
```

```
katia.up()  
katia.goto(-2, 10)
```

```
class Turtle:
```

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

```
    def up(self):  
        self.y += 1
```

```
    def goto(self, x, y):  
        self.x = x  
        self.y = y
```

```
    def get_position(self):  
        ...
```

```
    def print_position(self):  
        ...
```

Class

Object

Instantiate

Method

Attribute

Constructor

**self**



# Definitions

`katia is self`

**Inside Class**  
`self.attribute`  
`self.method`

## Outside Class

`katia.attribute`  
`katia.method`

```
katia = Turtle(0, 0)
```

```
katia.up()  
katia.goto(-2, 10)  
x, y = katia.get_position()
```

```
pint(x, y)  
>>> (-1, 10)
```

```
class Turtle:
```

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

```
    def up(self):  
        self.y += 1
```

```
    def goto(self, x, y):  
        self.x = x  
        self.y = y
```

```
    def get_position(self):  
        ...
```

```
    def print_position(self):  
        ...
```

Class

Object

Instantiate

Method

Attribute

Constructor

**self**

# Definitions

`katia is self`

**Inside Class**  
`self.attribute`  
`self.method`

## Outside Class

`katia.attribute`  
`katia.method`

```
katia = Turtle(0, 0)
```

```
katia.up()  
katia.goto(-2, 10)  
x, y = katia.get_position()
```

```
pint(x, y)  
>>> (-1, 10)
```

```
class Turtle:
```

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

```
    def up(self):  
        self.y += 1
```

```
    def goto(self, x, y):  
        self.x = x  
        self.y = y
```

```
    def get_position(self):  
        return self.x, self.y
```

```
    def print_position(self):  
        ...
```

Class

Object

Instantiate

Method

Attribute

Constructor

**self**

# Definitions

`katia is self`

**Inside Class**  
`self.attribute`  
`self.method`

## Outside Class

`katia.attribute`  
`katia.method`

```
katia = Turtle(0, 0)
```

```
katia.up()  
katia.goto(-2, 10)  
x, y = katia.get_position()
```

```
pint(x, y)  
>>> (-1, 10)
```

```
katia.print_position()  
>>> -1 10
```

```
class Turtle:
```

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

```
    def up(self):  
        self.y += 1
```

```
    def goto(self, x, y):  
        self.x = x  
        self.y = y
```

```
    def get_position(self):  
        return self.x, self.y
```

```
    def print_position(self):  
        ...
```

Class

Object

Instantiate

Method

Attribute

Constructor

**self**

# Definitions

`katia is self`

**Inside Class**  
`self.attribute`  
`self.method`

## Outside Class

`katia.attribute`  
`katia.method`

```
katia = Turtle(0, 0)
```

```
katia.up()  
katia.goto(-2, 10)  
x, y = katia.get_position()
```

```
pint(x, y)  
>>> (-1, 10)
```

```
katia.print_position()  
>>> -1 10
```

```
class Turtle:
```

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

```
    def up(self):  
        self.y += 1
```

```
    def goto(self, x, y):  
        self.x = x  
        self.y = y
```

```
    def get_position(self):  
        return self.x, self.y
```

```
    def print_position(self):  
        print(self.x, self.y)
```

Class

Object

Instantiate

Method

Attribute

Constructor

**self**

# Definitions

`katia is self`

**Inside Class**  
`self.attribute`  
`self.method`

```
class Turtle:
```

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

```
    def up(self):  
        self.y += 1  
        self.print_position()  
    def goto(self, x, y):  
        self.x = x  
        self.y = y  
        self.print_position()  
    def get_position(self):  
        return self.x, self.y
```

```
    def print_position(self):  
        print(self.x, self.y)
```

**Outside Class**

`katia.attribute`  
`katia.method`

```
katia = Turtle(0, 0)  
  
katia.up()  
katia.goto(-2, 10)  
x, y = katia.get_position()  
  
print(x, y)  
>>> (-1, 10)  
  
katia.print_position()  
>>> -1 10
```

Class

Object

Instantiate

Method

Attribute

Constructor

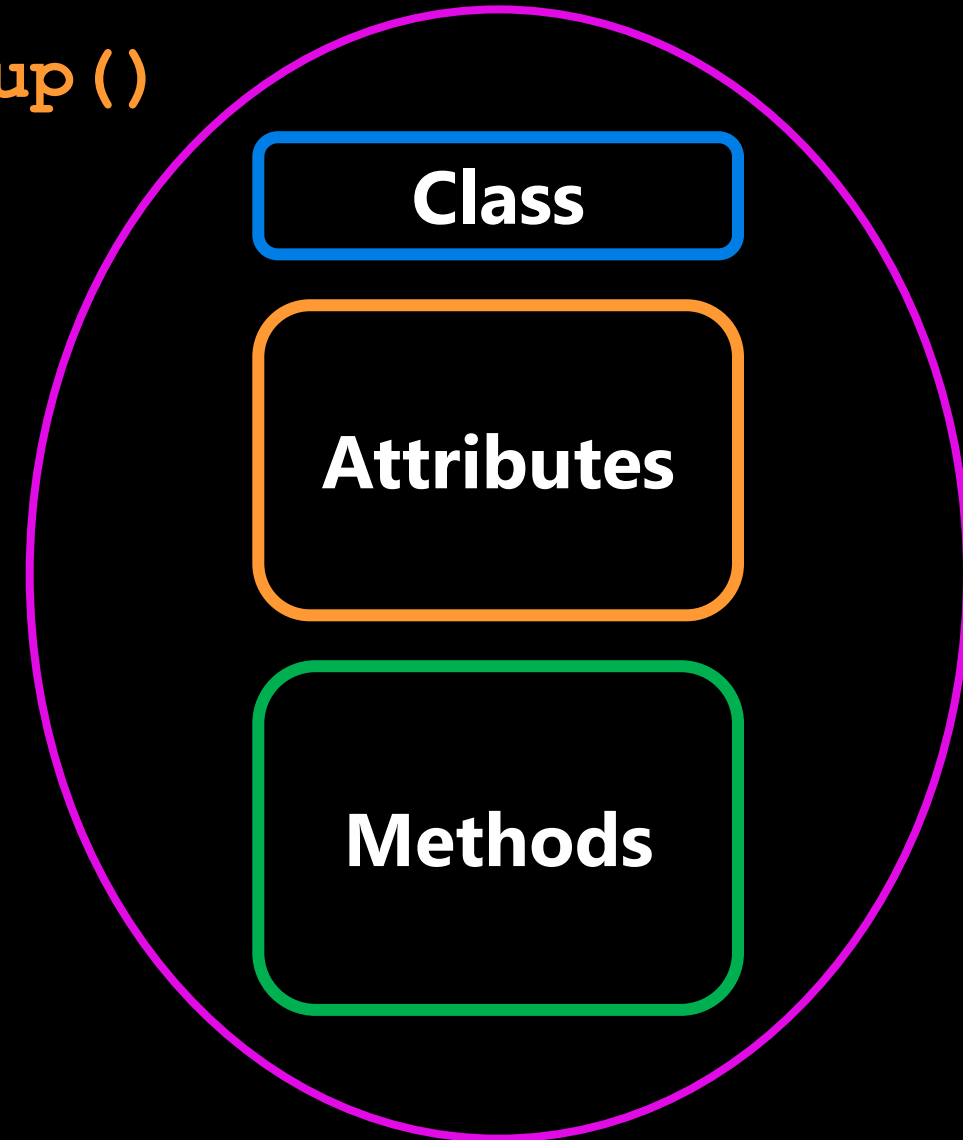
**self**

# Encapsulation

```
y = up(y)    alex.up()
```

- The core of object-oriented programming is the organization of the program by **encapsulating** related **data** and **functions** together in an object.
- To encapsulate something means to enclose it in some kind of container.
- In programming, encapsulation means keeping **data** and the **code** that uses it in one place and hiding the details of exactly how they work together.

## Encapsulation



# Point Class: Constructor

- Create a Point class to:
  - Contain data about the location of a Point instance.
  - Be able to calculate the distance between the Point instance and another point.

**Point**

**x**  
**y**

**distance between points**



# Point Class: Constructor

- Create a Point class to:
  - Contain data about the location of a Point instance.
- Let's start with the **attributes** and the **constructor**.

**Open your notebook**

**Click Link:**

**2. Write a Point Class: Constructor**

# Point Class: Methods

- Our Point class needs to:
  - Be able to calculate the distance between the Point instance and another point.
- Let's now write the **method**.

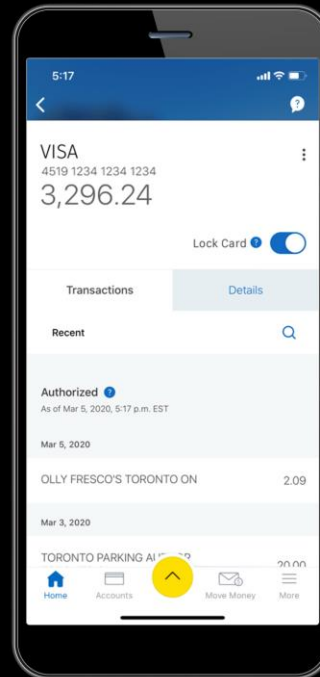
**Open your notebook**

**Click Link:**

**3. Write a Point Class: Methods**

# Encapsulation

- Let's highlight the value of encapsulation with a bank **Account** class.
- Attributes:
  - Account owner's name.
  - Current account balance.
- Methods:
  - Deposit money.
  - Withdraw money.
  - Print account balance.



**Open your  
notebook**

**Click Link:**  
**4. Bank Account  
Class**

## objects, classes, and methods.

Week 10 | Lecture 1 (10.1)

if nothing else, write `#cleancode`