APS106

UNIVERSITY OF TORONTO

objects, classes, and methods.

Week 6 | Lecture 1 (6.1.1)

While waiting for class to start:

Download and open the Jupyter Notebook (.ipynb) for Lecture 6.1.1

You may also use this lecture's JupyterHub link instead (although opening it locally is encouraged).

Upcoming:

- Reflection 6 released Friday @ 11 AM
- Lab 5 due this Friday
- Behrang's Coffee Break / Office Hours Friday @ 1 PM
- PRA (Lab) on Friday @ 2PM this week
- Exam Review - When is your preference?

if nothing else, write #cleancode

Today's Content

- Lecture 6.1.1
  - objects, classes, and methods
- Lecture 6.1.2
  - classes in classes!

Procedural vs Object-Oriented

- *"Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects", which may contain data, in the form of fields, often known as attributes; and code, in the form of procedures, often known as methods."*
  — Wikipedia
- *"Procedural programming is a programming paradigm, derived from structured programming, based upon the concept of the procedure call. Procedures, also known as routines, subroutines, or functions, simply contain a series of computational steps to be carried out."*
  — Wikipedia

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

Global Variable
Pedestrian 1
x, y Location

Global Variable
Pedestrian 2
x, y Location

Global Variable
Pedestrian 3
x, y Location

Global Variable
Traffic Light 1
Color

Global Variable
Car 1
x, y location

input

output

Function
OK to Cross

Function
Advance Position

Function
Pedestrian in Intersection

Object-Oriented Programming

Car Object
x, y Location
Pedestrian in Intersection
Advance Position

Traffic Light Object
Color
Change Color

Pedestrian Object
x, y Location
Ok to Cross
Advance Position

Pedestrian Object
x, y Location
Ok to Cross
Advance Position

Pedestrian Object
x, y Location
Ok to Cross
Advance Position

Encapsulation of Data and Functions

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

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

- **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.

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

>>> isinstance(max, object)
True

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

Objects in Python

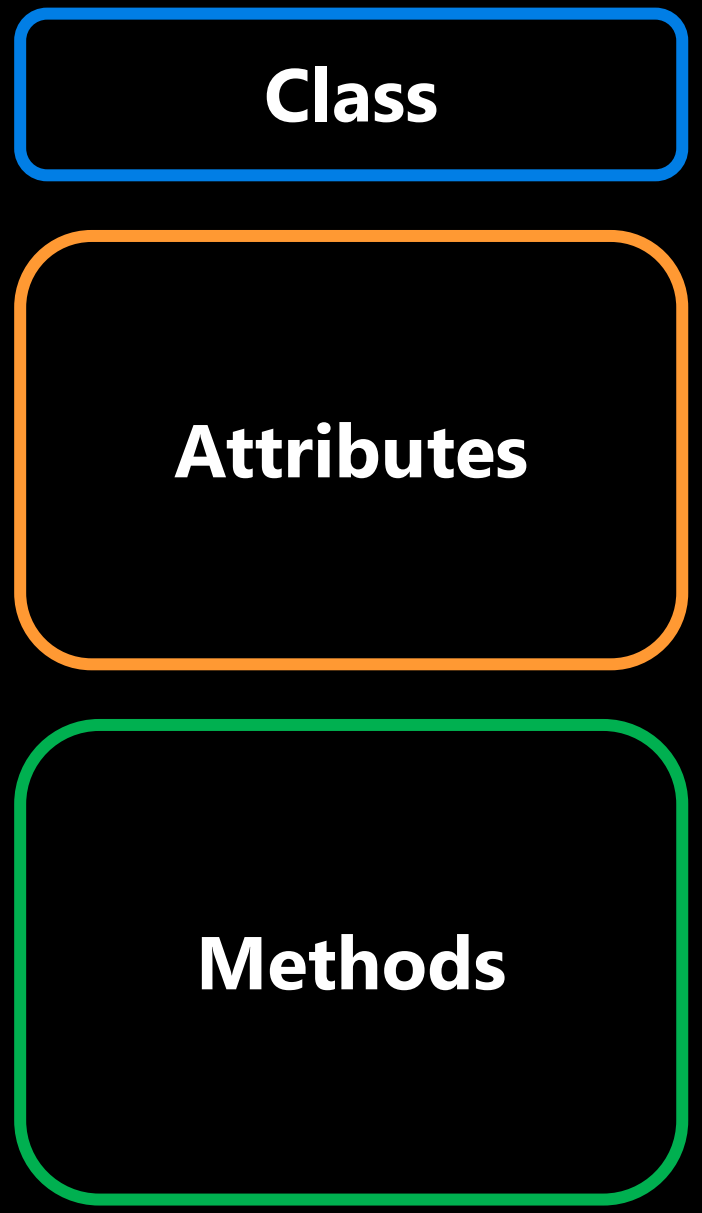
- Each object has a type or **class** it is associated with.

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
```

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**).



Classes

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

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

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

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

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

Class.

Object. Instance of the Person Class.

```
june = Person('June', 34,
              'Ottawa',
              'she/her')
```

```
type(june)
>>> Person
```

Class.

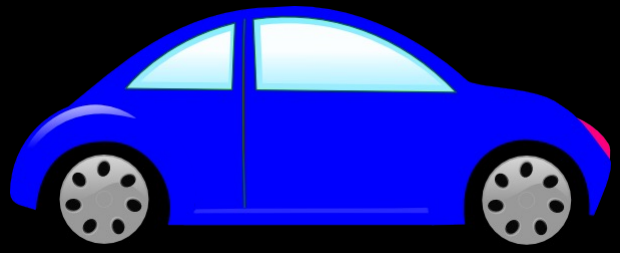
Person

name
age
city
gender

eat
study
sleep
play

Classes

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



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

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

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

Car

model
company
year
color

brake
accelerate
change oil
open trunk

Classes

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

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

Class.

Object. Instance of the Car Class.

```
mycar = Car('Corolla',
            'Toyota',
            1980,
            'red')
```

```
type(mycar)
>>> Car
```

Class.

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: Brian
x location: 92
y location: 62

name: Vinodh
x location: 134
y location: 45

Turtle

name
x location
y location

move up
move down
move left
move right
go to

Classes

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

name: Lucy
x location: 24
y location: 35

Object. Instance of the Turtle Class.

Class.

```
turtle = Turtle('Lucy',
                24,
                35)

type(turtle)
>>> Turtle
```

Class.

Turtle

name
x location
y location

move up
move down
move left
move right
go to

Classes

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

name: bills.txt
mode: append

name: itinerary.txt
mode: write

name: grades.txt
mode: read

File

name
mode

read
readline
readlines

Classes

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

name: bills.txt
mode: append

Object. Instance of the File Class.

Class.

```
myfile = File('bills.txt')

type(myfile)
>>> File
```

Class.

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

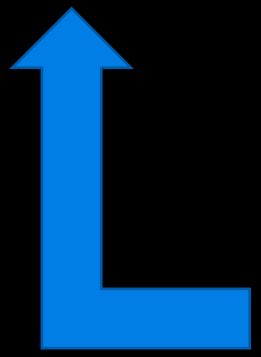
Constructor

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

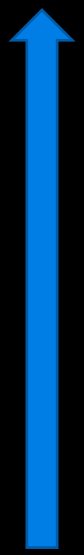
- A function defined in a class.

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

```
class Turtle:

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

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



Function

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** 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

```
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
```

```
katia = Turtle(0, 0)
print(katia.x, katia.y)
```

```
joseph = Turtle(-4, 1)
print(joseph.x, joseph.y)
```

Class

Object

Instantiate

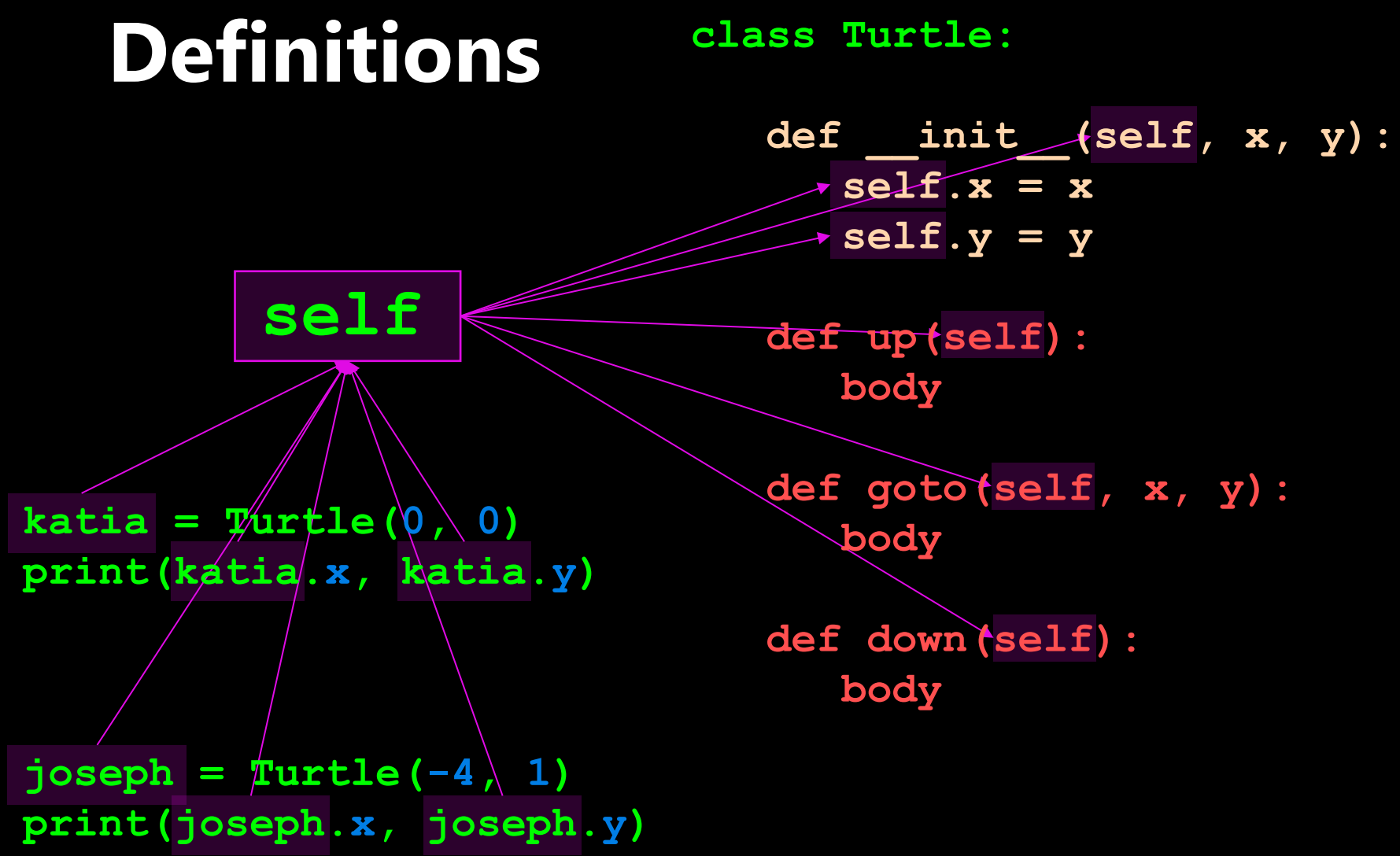
Method

Attribute

Constructor

self

Definitions



Class

Object

Instantiate

Method

Attribute

Constructor

self

Definitions

katia is self

Inside Class
self.attribute
self.method

Outside Class
katia.attribute
katia.method

```
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):
        ...
```

```
katia = Turtle(0, 0)

katia.up()
```

Class
Object
Instantiate
Method
Attribute
Constructor
self

Definitions

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

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

`katia is self`

```
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):
        ...
```

```
katia = Turtle(0, 0)

katia.up()
```

Class

Object

Instantiate

Method

Attribute

Constructor

self

Definitions

katia is self

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

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

```
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):
        ...
```

```
katia = Turtle(0, 0)

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

Class

Object

Instantiate

Method

Attribute

Constructor

self

Definitions

katia is self

Inside Class
self.attribute
self.method

Outside Class
katia.attribute
katia.method

```
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):
        ...
```

```
katia = Turtle(0, 0)

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

Class

Object

Instantiate

Method

Attribute

Constructor

self

Definitions

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

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

`katia is self`

```
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):
        ...
```

```
katia = Turtle(0, 0)

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

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

Class

Object

Instantiate

Method

Attribute

Constructor

self

Definitions

katia is self

Inside Class
self.attribute
self.method

Outside Class
katia.attribute
katia.method

```
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):
        ...
```

```
katia = Turtle(0, 0)

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

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

Class
Object
Instantiate
Method
Attribute
Constructor
self

Definitions

katia is self

Inside Class
self.attribute
self.method

Outside Class
katia.attribute
katia.method

```
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):
        ...
```

```
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

Definitions

katia is self

Inside Class
self.attribute
self.method

Outside Class
katia.attribute
katia.method

```
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)
```

```
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

Definitions

katia is self

Inside Class
self.attribute
self.method

Outside Class
katia.attribute
katia.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)
```

```
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

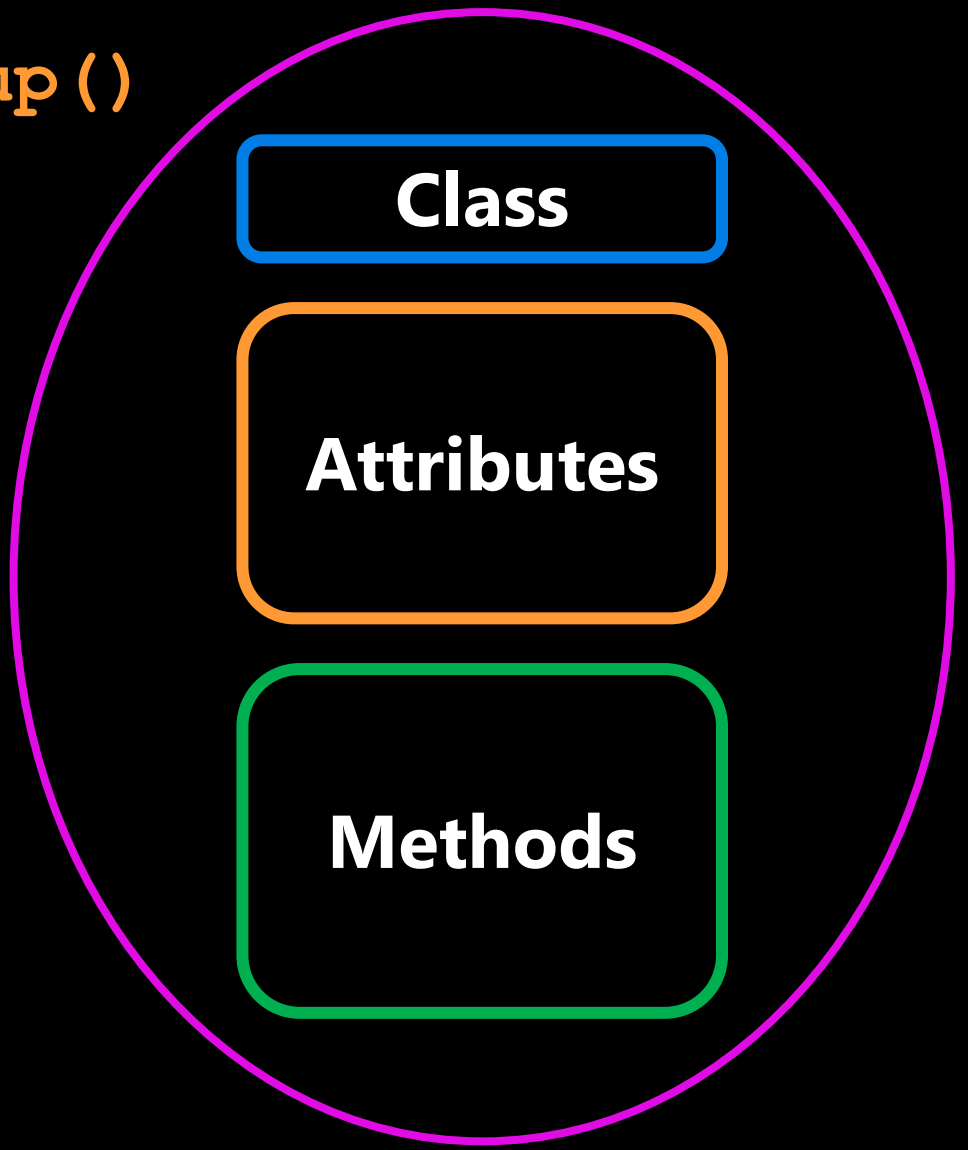
Constructor

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.



Point Class: Constructor

- The real strength of object-oriented programming comes from being able to define new classes.
- In math, in two dimensions, a point is two numbers (coordinates) that are treated collectively as a single object.
  - For example, (0, 0) represents the origin, and (x, y) represents the point **x** units to the right and **y** units up from the origin.
- Its also common to want to calculate the distance between two points.

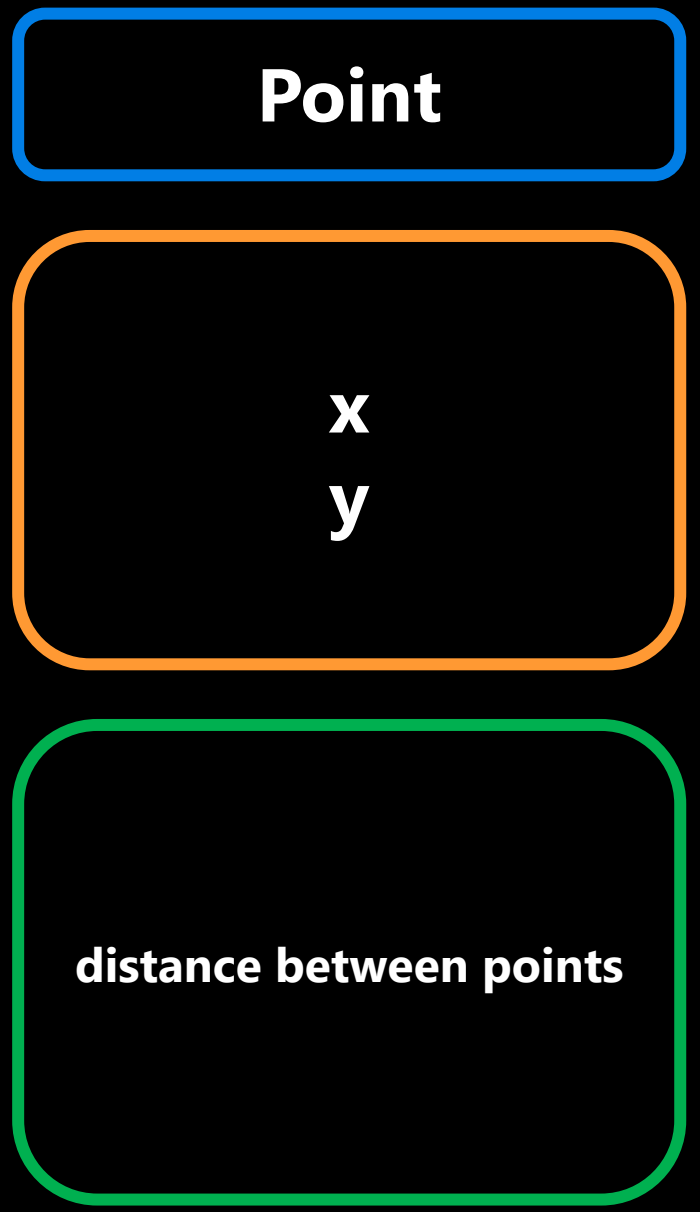
Class

Attributes

Methods

Point Class: Constructor

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



Point Class: Constructor

- Our Point class needs 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

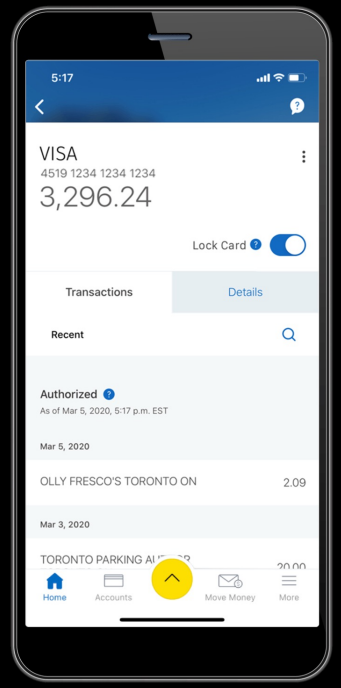
- Our Point class needs to:
  - Be able to calculate the distance between the Point instance and another point.
- Let's now write the method.

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

APS106

UNIVERSITY OF TORONTO

objects, classes, and methods.

Week 6 | Lecture 1 (6.1.1)

if nothing else, write #cleancode