CEGE0096: Geospatial Programming

Lecture 4: Object-Oriented Programming

October 21, 2019

Aldo Lipani

IMPERATIVE PROGRAMMING (RECAP)

Our Python (Imperative) Journey

- Types
- Operators
- Variables
- Simple I/O
- Branching

} Covered in the 1st Lecture

- Loops
- Functions
- Program Planning

} Covered in the 2nd Lecture

- Strings
- Tuples
- Lists
- Dictionaries
- Files

} Covered in the 3rd Lecture

Review Exercise 5

Given a sequence of numbers inputted by the user compute the following statistics:

- min,
- max,
- average,
- number of duplicates, and
- mode.

You should define a function for each one of these statistics.

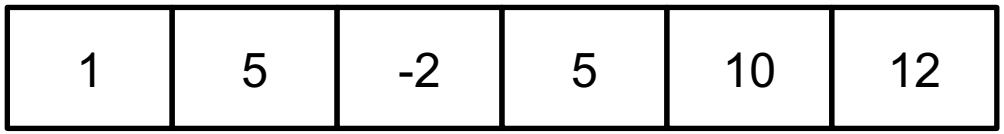
TIP: these functions should take as input a sequence of numbers.

Solution

```
print("Describe My Sequence (0.1)")

# functions go here

n = int(input("How many values? "))
l = []
for i in range(n):
    v = float(input("Insert number " + str(i + 1) + ": "))
    l.append(v)



print("The statistics are:")
print("* Min:", min(l))
print("* Max:", max(l))
print("* Avg:", average(l))
print("* Dup:", duplicates(l))
print("* Mod:", mode(l))
```

Solution (I)

```
def min(vs):
    res = vs[0]
    for v in vs[1:]:
        if v < res:
            res = v
    return res
```



Solution (II)

```
def max(vs):
    res = vs[0]
    for v in vs[1:]:
        if v > res:
            res = v
    return res
```

1	5	-2	5	10	12

Solution (III)

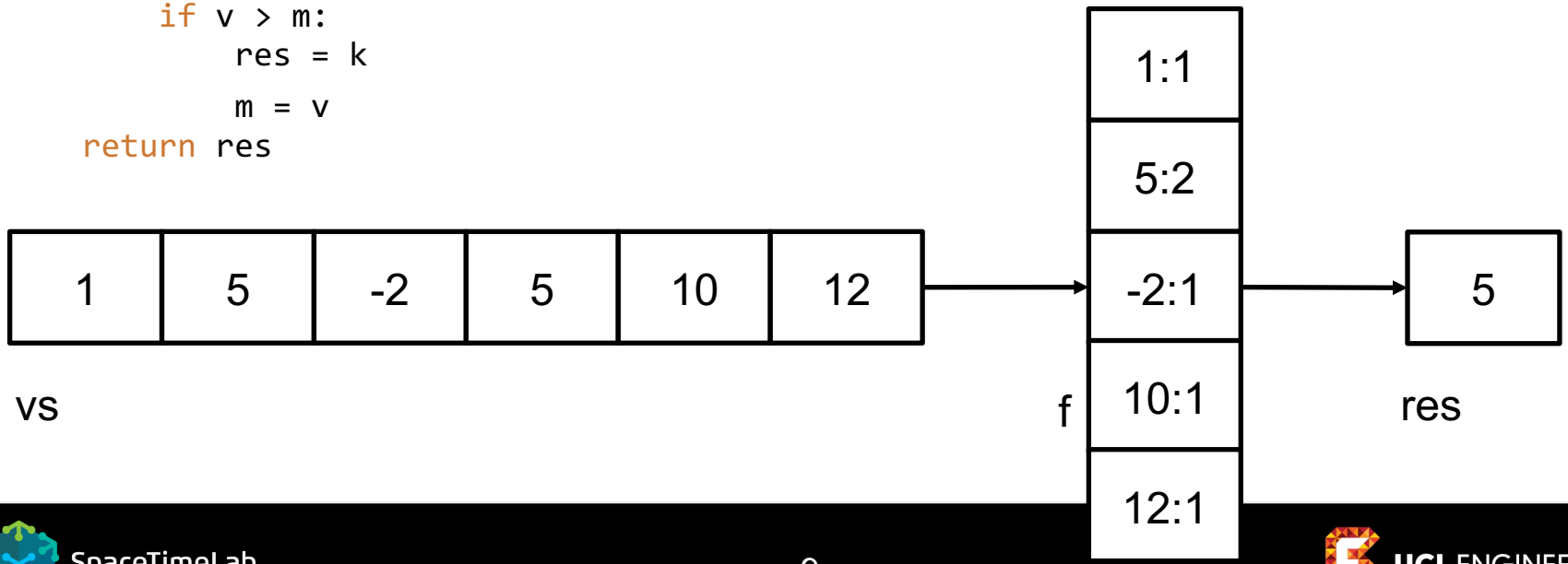
```
def average(vs):
    res = 0
    for v in vs:
        res = res + v
    res = res / len(vs)
    return res
```

Solution (IV)

```
def duplicates(vs):
    s = set()
    res = 0
    for v in vs:
        if v not in s:
            s.add(v)
        else:
            res = res + 1
    return res
```

Solution (V)

```
def mode(vs):
    f = {}
    for v in vs:
        if v not in f:
            f[v] = 0
        f[v] = f[v] + 1

    res = vs[0]
    m = f[res]
    for k, v in f.items():
        if v > m:
            res = k
            m = v
    return res
```



Review Exercise 6

Write a program that **asks** the user to `input` a temperature in degree Celsius and `print`s out a suitable message according to the temperature state below:

- "Freezing" when the temperature is below t_1;
- "Very cold" when it's between t_1 and t_2 (not included);
- "Cold" when it's between t_2 and t_3 (not included);
- "Normal" when it's between t_3 and t_4 (not included);
- "Hot" when it's between t_4 and t_5 (not included);
- "Very hot" when it's equal or above t_5.

The temperatures t_1, t_2, t_3, t_4 and t_5 are given as input by the user.

Solution

```
print("Temperature to Perceived Temperature Converter (0.3)")

perceived_temps = ["freezing", "very cold", "cold", "normal", "hot", "very hot"]
temps = []
for i in range(5):
    temp = float(input("To which temperature you feel like is no longer " +
                       perceived_temps[i] + "? "))
    temps.append(temp)

temp = float(input("What temperature is outside? "))
perceived_temp = perceived_temps[-1]
not_found = True
i = 0
while not_found and i < len(temps):
    if temp < temps[i]:
        perceived_temp = perceived_temps[i]
        not_found = False
    i = i + 1

print("Your temperature today is", perceived_temp)
```

```
for i in range(len(temps)):
    if temp < temps[i]:
        perceived_temp = perceived_temps[i]
        break
```

OBJECT-ORIENTED PROGRAMMING

Our Second Python (OOP) Journey

- Class
- Object
- Attribute
- Method
- Inheritance
- Polymorphism
- Encapsulation

Why Object-Oriented Programming?

Object-oriented programming (OOP) derives from the **software crisis** of the 60's:

"The major cause of the software crisis is that the machines have become several orders of magnitude more powerful!"

~Edsger Dijkstra, The Humble Programmer

Problem: the overall complexity of hardware and the software development process was increasing.

Indicators:

- Projects running over budget or over time
- Inefficiency and low quality of software
- Failure of software to meet the requirements (including delivery)
- Unmanageable projects and unmaintainable code

Object-Oriented Programming

Imperative programming (non-Object-oriented) method can cause software to be extremely **difficult to maintain**.

Code is **hard to reuse** in other projects.

OOP is a programming paradigm that attempts to address these issues by structuring the code into classes.

Some Key Terms in OOP

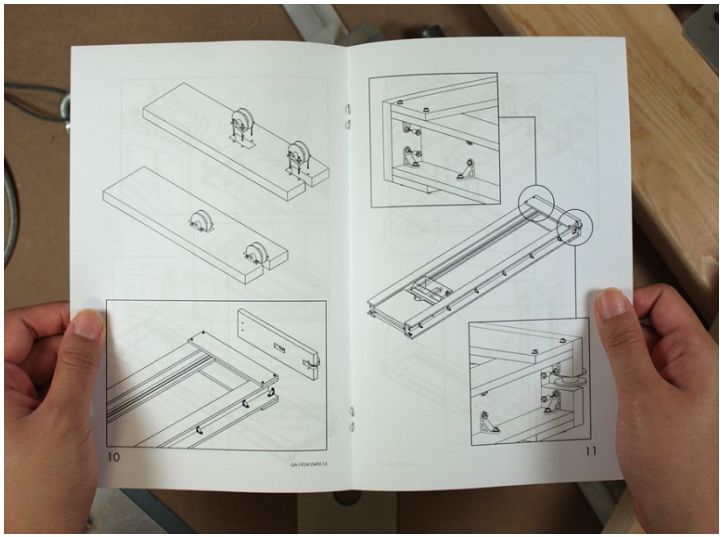
A **class** is a template for making *objects*, containing *attributes* and *methods*.

An **object** is an instance (or concretization) of a class.

An **attribute** is a variable that belongs to an object. Attributes define the state of an object.

A **method** is a function that belongs to an object that operates with the attributes.

Classes vs. Objects



Class: Defines the way in which an object should be constructed.

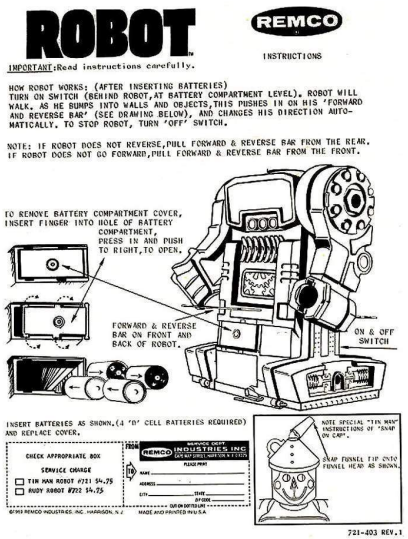
Object: It is a single instance of the class, constructed using the class definition.

We can change (or perhaps break) an object without changing the class (i.e. we can put the legs on the chair the wrong way but the instructions don't change).

If the class is well-defined, the object should be hard to break (to not be used as designed).

Attributes vs. Methods

Class: Robot



Object: Sparky

Attributes:

- height
- number of arms
- battery life

Methods:

- walk()
- speak()
- shoot()

Typical usage

```
sparky = Robot(50)   # Create an instance of the robot class with height 50
sparky.shoot()       # Use the shoot method
"Pew pew pew!"       # Perhaps the shoot method outputs this string
```

You Already Worked with Objects

Examples are data structures:

- A `set` is a class which has attributes and methods.
- When you reference a set in your code, you are creating a set object.

Create an instance of a set:

```
s = set()
```

Add is a method of the class set that changes the internal state of the set:

```
s.add(10)
```

You've worked with classes, objects, attributes, and methods; now you will create them.

A Class in Python

```
class Robot:

    def __init__(self, name):                    # Constructor
        self.name = name                         # Attributes
        self.ammunition = 99

    def shoot(self):                             # Method
        if self.ammunition > 0:
            print("Pew pew pew!")
            self.ammunition = self.ammunition - 3
        else:
            print("No pew...")
```

Basic Class

`self` (`this` in Java and C++) is the first parameter of every method inside a class.

The attributes of a class are stored locally in every instance (object).

`self` is the way that methods can access data stored within the object.

`super()` is used to refer to the parent class. Commonly seen when redefining the constructor of a sub-class.

OOP Features

Inheritance. The ability to create a child (sub-)class from a parent (super-)class inheriting its attributes and methods. This enables hierarchical data structures and avoids code repetition.

Polymorphism. The ability to override methods of a parent class in a child class, changing how methods work depending on the context.

Encapsulation. Access to attributes and methods is under the control of the programmer (e.g. “public” or “private” attributes; “public” or “private” methods)

Inheritance

One of the most useful features of OOP is the ability of classes to inherit attributes and methods from a parent (or super) class.

Defining common properties and methods in a parent class allows child (sub) classes to have access to them without repetition of code.

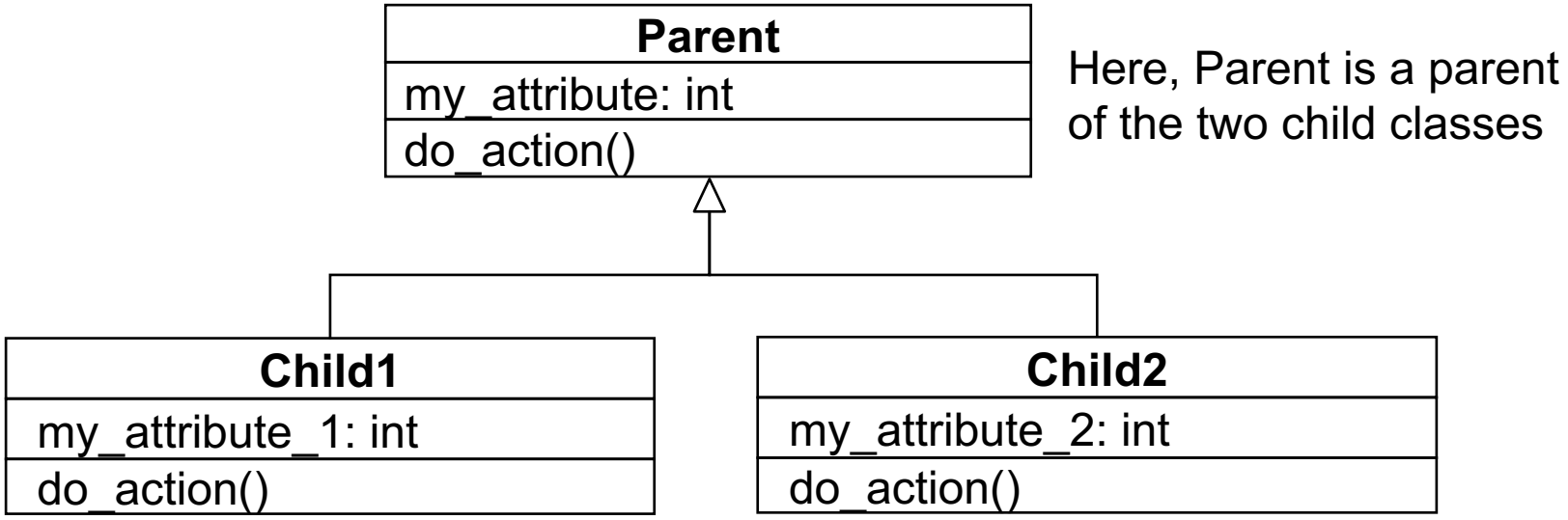
Inheritance can reduce the overall size of a program and make it much easier to debug code:

- Fixes in a parent class will apply to all child classes automatically;
- Problems with a child class will not affect its parent class.

Inheritance also allows neat hierarchical representations of some real world phenomena and processes.

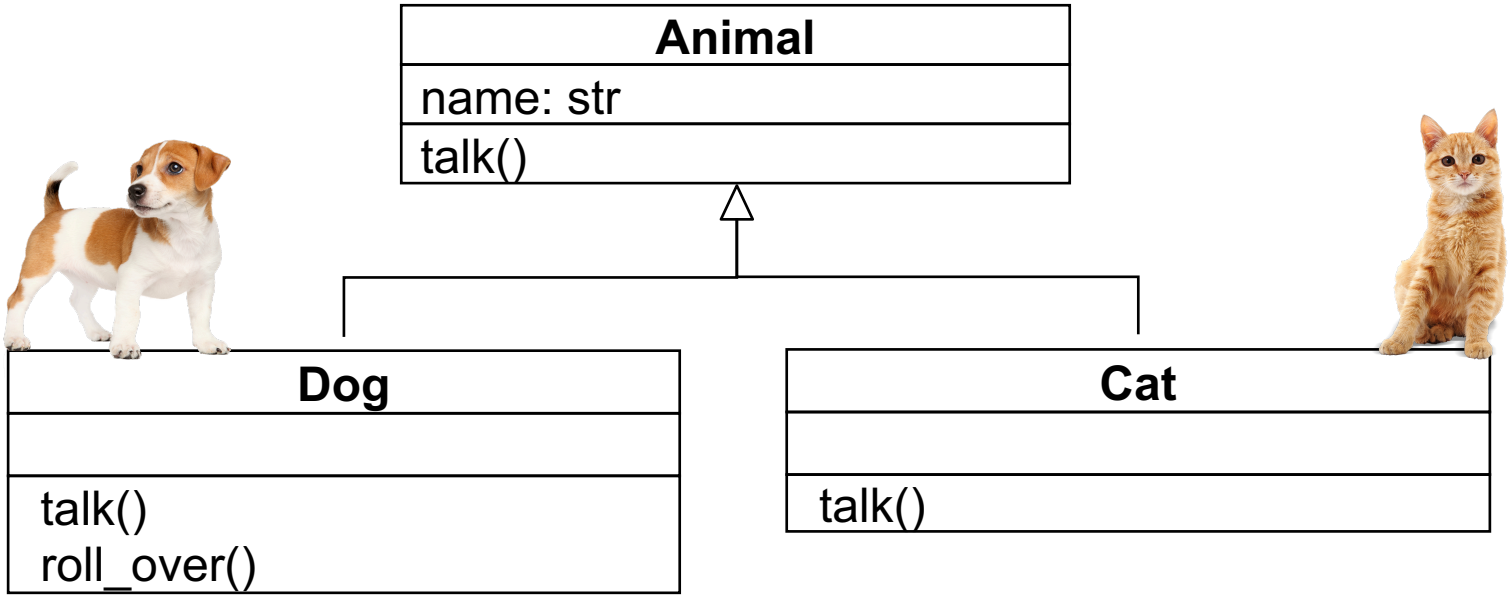
Inheritance: Class Diagram

A Unified Modeling Language (UML) **class diagram** is a type of static structure diagram that describes the structure of a computer program. It shows the program's classes, their attributes and methods, and their parent-child relationships with one another.



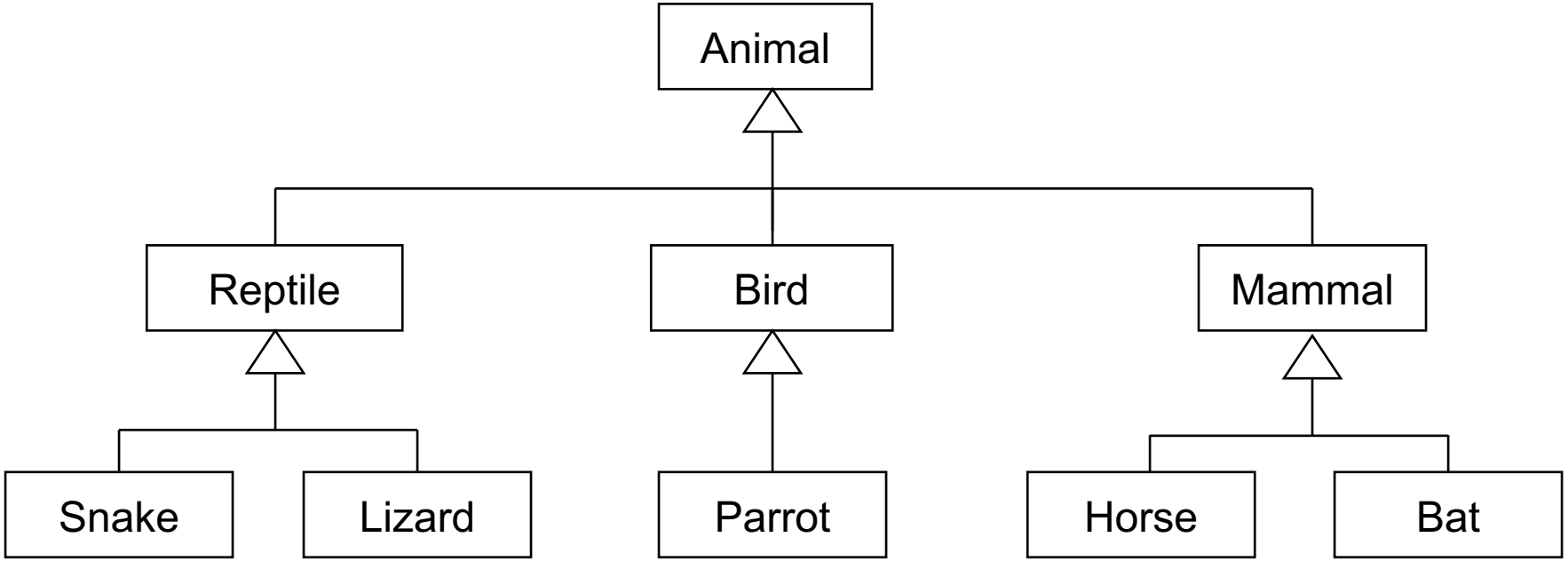
Here, Parent is a parent of the two child classes

Example: Animal Class



Cats and dogs are both animals, so share some common properties and methods.

Animal Class Diagram



Inheritance: How is it done?

```
class Animal:

    def __init__(self, name):
        self.name = name

    def talk(self):
        print("I don't know how to talk!")


class Dog(Animal):

    def talk(self):
        print("Woof!")


class Cat(Animal):

    def talk(self):
        print("Meow!")
```

Multiple Inheritance

A class can inherit from multiple parent classes.

```
class Animal:

    def __init__(self, name):
        self.name = name

    def talk(self):
        print("I don't know how to talk!")


class Mammal(Animal):

    def milk(self):
        print("I don't know which milk!")


class Bat(Mammal, Winged):

    def talk(self):
        print("Bat-voice!")

    def milk(self):
        print("Bat milk")
```

```
class Winged(Animal):

    def fly(self):
        print("Flap flap flap!")
```

Polymorphism

Polymorphism allows the same function or method to be used in different ways depending on the context.

In OOP, polymorphism is often used to override methods of a parent class in a child class.

- e.g. the talk methods in the animal examples
- It also allows functions to operate differently depending on the data supplied to them.
  - E.g. the + operator functions differently depending on integer, string and float input

```
print("Hello" + "There")  # Two strings
print(1 + 2)              # Two integers
print(["A", "B"] + ["C"]) # Two lists
```

Polymorphism

```
class Animal:

    def __init__(self, name):
        self.name = name

    def talk(self):
        print("I don't know how to talk!")



def talking_animal(animal):
    animal.talk()

talking_animal(Cat("Fluffy"))
talking_animal(Dog("Spot"))
```

Encapsulation

Encapsulation describes the idea of using methods to manipulate the data within an object.

This concept is also often used to hide the internal representation, or state, of an object from the outside.

So far we have used only public attribute and methods. This means that these are accessible from outside.

You can disallow this access by using private attributes and methods.

Encapsulation

Encapsulation is used to control the way in which users interact with the properties and methods within a class.

You can use a double underscore (__) to make an attribute or a method **private**.

```
class Animal:

    def __init__(self, name):
        self.__name = name                    # This attribute is now private

    def __method(self):                       # This method is private
        print("This method can be called only by myself!")

    def talk(self):
        print("I don't know how to talk!")
```

Private Attributes

If an attribute is private this cannot be accessed from outside or by a child class.

To access these attributes it is common to use **getter** and **setter** methods, based on whether you want to provide read or write rights or have some control over these operations.

```
class Animal:

    def __init__(self, name):
        self.__name = name

    def get_name(self):
        return self.__name

    def set_name(self, name):
        self.__name = name
```

Summary

OOP has advantages over procedural programming:

- Less code repetition;
- Easier debugging.

Centered around objects, which are instances of **classes**.

Classes contain **attributes** (variables) and **methods** (functions).

Key features: **inheritance**, **polymorphism**, and **encapsulation**.

Well suited to hierarchical data modeling.

Exercise 7 (30 Minutes)

Design an object-oriented program for the following classes:
`Geometry`, `Point`, `Line`, `Polygon`, `Triangle` and `Square`.

This program should allow the construction of:

- A point;
- A line given two points;
- A polygon given 3 or more points;
- A triangle given 3 points;
- A square given its bottom left point and its side length.

These classes should implement the methods to get or compute (where and when appropriate): the name, the x, the y, the lines, and the area.

TIP: use the Shoelace Formula to compute the are of a triangle given 3 points.

Solution

```
class Geometry:

    def __init__(self, name):
        self.__name = name

    def get_name(self):
        return self.__name


class Point(Geometry):

    def __init__(self, name, x, y):
        super().__init__(name)
        self.__x = x
        self.__y = y

    def get_x(self):
        return self.__x

    def get_y(self):
        return self.__y
```

Solution (II)

```
class Line(Geometry):

    def __init__(self, name, point_1, point_2):
        super().__init__(name)
        self.__point_1 = point_1
        self.__point_2 = point_2


class Polygon(Geometry):

    def __init__(self, name, points):
        super().__init__(name)
        self.__points = points

    def get_points(self):
        return self.__points

    def lines(self):
        res = []
        points = self.get_points()
        point_a = points[0]
        for point_b in points[1:]:
            res.append(Line(point_a.get_name() + "-" + point_b.get_name(), point_a, point_b))
            point_a = point_b
        res.append(Line(point_a.get_name() + "-" + points[0].get_name(), point_a, points[0]))
        return res
```

Solution (III)

```
class Triangle(Polygon):

    def __init__(self, name, point_1, point_2, point_3):
        super().__init__(name, [point_1, point_2, point_3])

    def area(self):
        res = 0
        ps = self.get_points()
        # Shoelace Formula
        res = res + ps[1].get_x() * ps[2].get_y() - ps[2].get_x() * ps[1].get_y()
        res = res - ps[0].get_x() * ps[2].get_y() + ps[2].get_x() * ps[0].get_y()
        res = res + ps[0].get_x() * ps[1].get_y() - ps[1].get_x() * ps[0].get_y()
        res = abs(res) / 2
        return res
```

Solution (IV)

```
class Square(Polygon):

    def __init__(self, name, bl_point, side):
        points = [bl_point,
                  Point("top left point", bl_point.get_x(), bl_point.get_y() + side),
                  Point("top right point", bl_point.get_x() + side, bl_point.get_y() + side),
                  Point("bottom right point", bl_point.get_x() + side, bl_point.get_y())]
        super().__init__(name, points)
        self.__side = side

    def area(self):
        return self.__side ** 2
```

Solution (V)

```
point_a = Point("A", 0, 0)
point_b = Point("B", 0, 1)
point_c = Point("C", 1, 1)

triangle = Triangle("A'", point_a, point_b, point_c)

print("Lines of the triangle " + triangle.get_name() + ":")
for line in triangle.lines():
    print(line.get_name())

print("Area:", triangle.area())
```

Exercise 8 (10 Minutes)

Write a program that asks the user to `input` a temperature in degree Celsius and `print`s out a suitable message according to the temperature state below:

- "Freezing" when the temperature is below t_1;
- "Very cold" when it's between t_1 and t_2 (not included);
- "Cold" when it's between t_2 and t_3 (not included);
- "Normal" when it's between t_3 and t_4 (not included);
- "Hot" when it's between t_4 and t_5 (not included);
- "Very hot" when it's equal or above t_5.

The temperatures t_1, t_2, t_3, t_4 and t_5 are **stored in a file named `temps.csv` like this:**

```
0
10
20
30
40
```

Solution

```
print("Temperature to Perceived Temperature Converter (0.4)")

perceived_temps = ["freezing", "very cold", "cold", "normal", "hot", "very hot"]
temps = []
with open("temps.csv", "r") as f:
    for line in f.readlines():
        temp = float(line)
        temps.append(temp)

temp = float(input("What temperature is outside? "))
perceived_temp = perceived_temps[-1]
for i in range(len(temps)):
    if temp < temps[i]:
        perceived_temp = perceived_temps[i]
        break

print("Your temperature today is", perceived_temp)
```

See you on Thursday for the Practical

Christopher Ingold Building G20

Do not forget to take your personal laptop with you.