

Welcome.

Everyone:

- Pull the updates from the course GitHub repo:
 - `cd <46120-PiWE repo>`
 - `git pull upstream main` ← you might have “upstream2” instead

Physical students:

- Sit WHEREVER you want. 😎
- Turn off laptop volume (mute). **←IMPORTANT!**
- Log into the Zoom meeting.
 - Microphone muted. Camera off.



46120: Scientific Programming for Wind Energy

Classes, inheritance and OOP (Object Oriented Programming)

Ju Feng



Agenda for today

- Pull new course material ✓
- Round robin
- Classes, inheritance and OOP
- Begin homework



Homework in last week

- **PART 0: Sign up for a final project team (ASAP)**

- Make sure you have done so if you want to pass this course
- Form teams of 2 to 3. !!!DELETE!!! Your name at left when you sign up for a team. Anyone not in team by deadline in announcement will receive "EM" (no-show) grade in course. (Check the sign up excel sheet)

- **PART 1: Explore one of the pre-defined projects**

- Discuss with your group, choose one pre-defined project to explore.
- Write a function to load, parse and plot the provided dataset in inputs, generate at least one figure. Ideally also with test in tests.
- Think about the structure/architecture of your package.

Note: the project you choose in this homework doesn't mean you have to stick to that project. Your group can make decision later on which project to work on. If you have a better project idea that your group want to pursue, first check the section on Custom project, and then come talk to us.

But remember to make a decision early, so that your group has enough time to work on it.



Round robin

Share solutions with your peers and give feedback.



Time to review and collaborate.

- 1 round of 25 minutes.
- 5 minutes: chaos.
- 20 minutes: present/discuss homework. Today's feedback focus:
 1. How is the other team's code? Easy to understand? Do the required job?
 2. How do you think about the other team's plan for the structure/architecture of their package? Do you have any suggestions? Can you learn something from them?
 3. Any other challenges you found?

- Afterwards: plenum discussion.
 - Be ready with questions!

PLENUM AT 09.30

Week 9

=====

BOR 0: StopFuckingSpiders, Lightning_McTeam, LetsGoRepo

BOR 1: WindFusion, Push-Pray, brunchyy

BOR 2: Kala-Krasia, bug_hunters, Let_Me_Help_You

BOR 3: SOUL-FINDERS, NoOneKnows, la_bombas_del_diablo

BOR 4: Pythonagoras, Grustlers, Breeze-Tech

BOR 5: WindWizards, Mouxlin AE, BladePYRunner

BOR 6: FatalError2, Git_Happens, WIndWise

BOR 7: WindCoderss, WindGPT, codingteam

BOR 8: Los-Programadores, Crypto-Mania

Notes in plenum.



Start at ...

Classes, inheritance and OOP



Background

- What is the biggest challenge in software engineering?

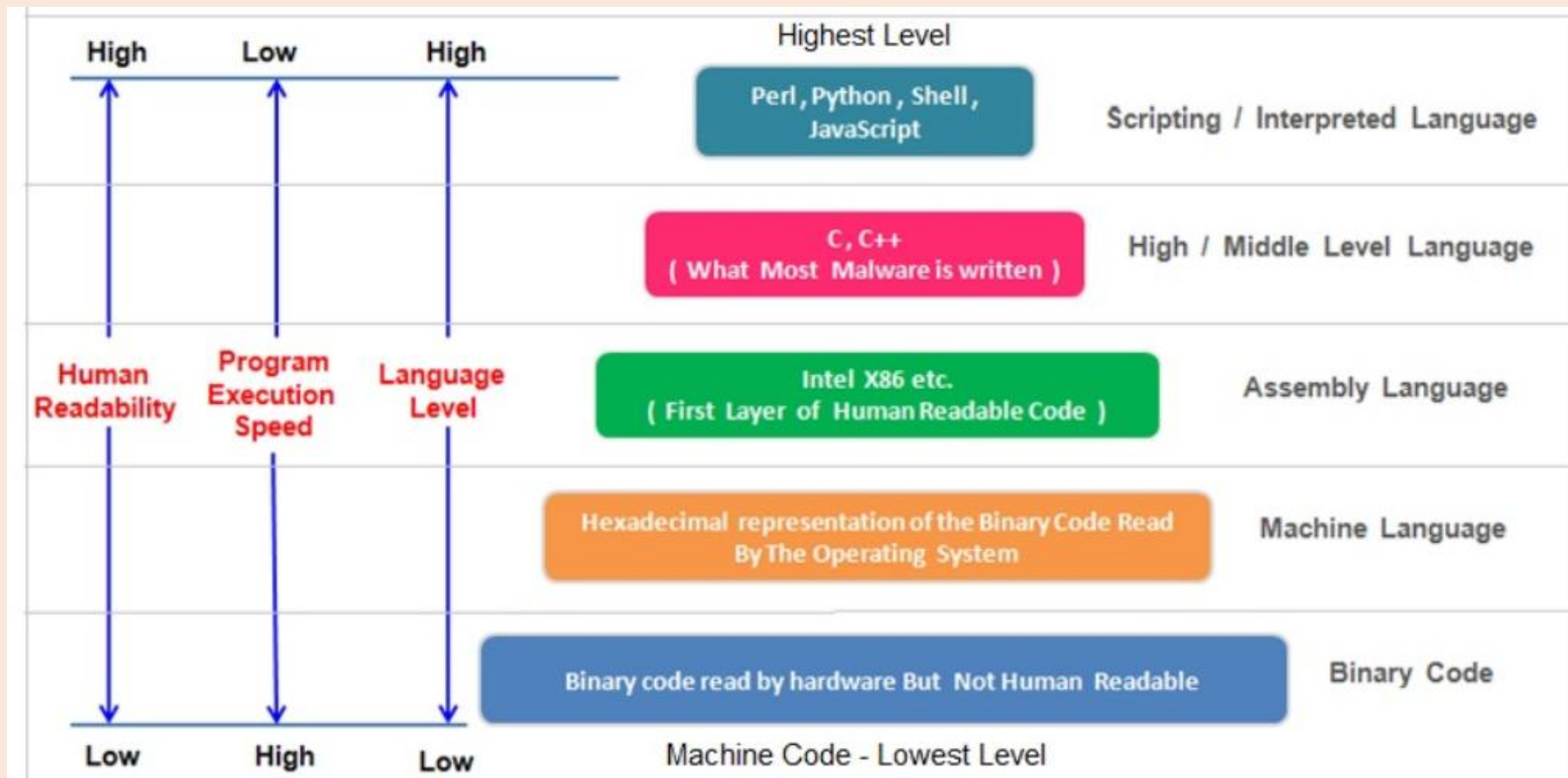
Managing complexity is the most important technical topic in software development.

-- Steve McConnell in *Code Complete*

- OOP (object-oriented programming) is an important programming concept/method, also called programming paradigm, that can help a lot in managing complexity.

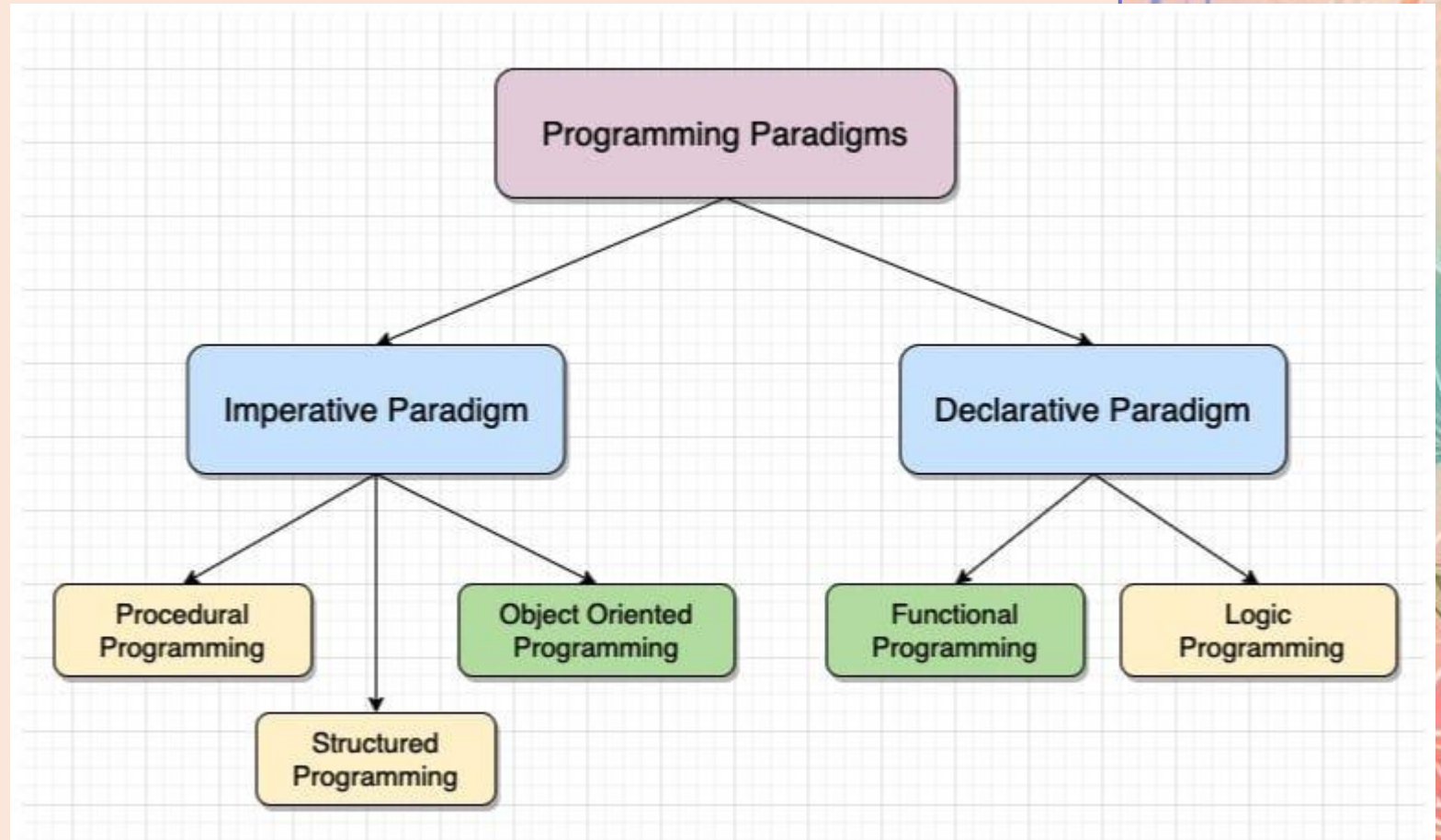


Levels of programming languages



Programming paradigms

- “A programming paradigm is a fundamental approach or style of programming that provides a set of principles, concepts, and techniques for designing and implementing computer programs. It defines the structure, organization, and flow of the code, as well as the methodologies for problem-solving and expressing computations.”



Source: Ario Liyan, (2023) What is a programming paradigm?

[link: <https://medium.com/@Ariobarxan/what-is-a-programming-paradigm-ec6c5879952b>]

What is OOP?



what is OOP



OOP stands for Object-Oriented Programming, which is a programming paradigm that organizes software design around the concept of objects. Objects are instances of classes, which encapsulate data and behavior (methods) that operate on that data.

OOP is based on the principles of abstraction, encapsulation, inheritance, and polymorphism.

- Abstraction allows us to focus on essential features of an object and hide irrelevant details.
- Encapsulation helps in keeping the data and methods together and secure from unauthorized access by external code.
- Inheritance allows creating new classes based on existing ones, inheriting their properties and methods.
- Polymorphism means that objects of different classes can be treated as if they are of the same class, making the code more flexible and reusable.

OOP provides a way to organize code in a more modular, reusable, and maintainable way. It has become the dominant programming paradigm in the software industry, and it is used in many programming languages such as Java, Python, and C++.

- OOP stands for Object-Oriented Programming, which is a **programming paradigm** that organizes software design around the concept of **objects**. Objects are instances of classes, which encapsulate data and behavior (methods) that operate on that data.

Source: <https://chat.openai.com/chat>



What is OOP?

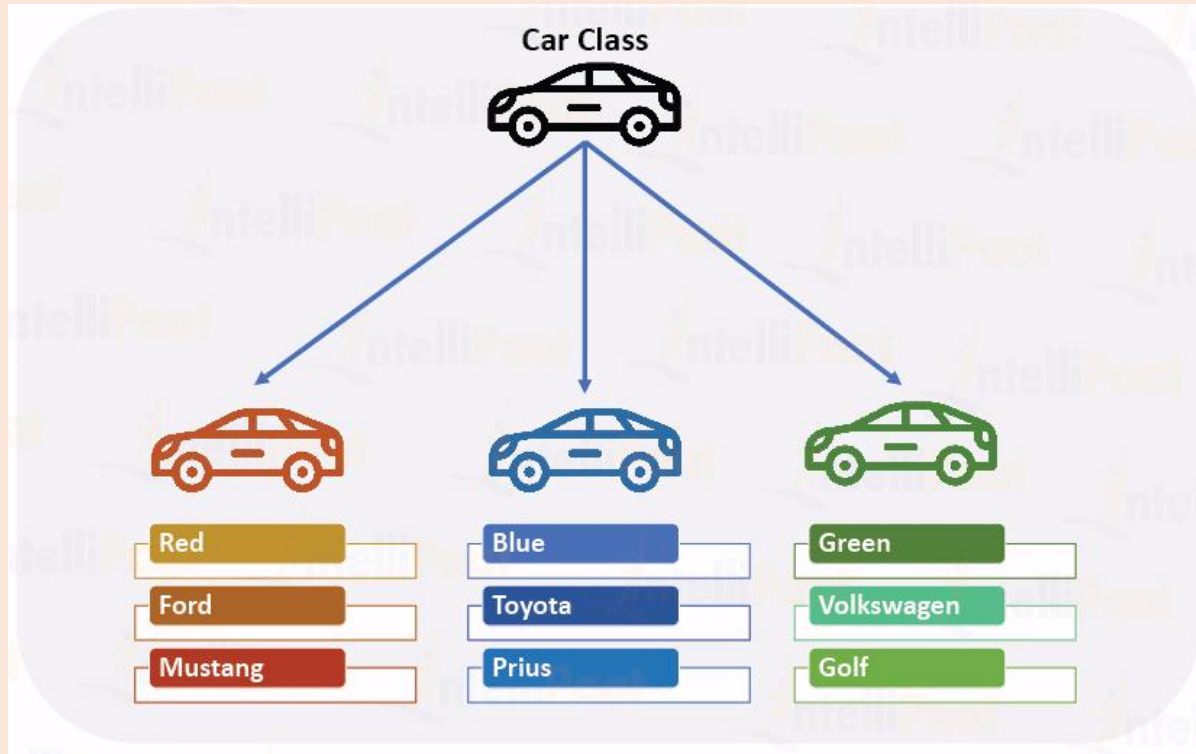


Figure source: <https://intellipaat.com/blog/tutorial/python-tutorial/python-classes-and-objects/>

- Something is an object (instance) of a type (class) of things.
- A class of things can be commonly characterized by a number of attributes (properties), and can do a number of things (methods).
- Different objects can have different values for the same attributes and do the same thing differently.
- A Class is a blueprint of a group of objects.
- A Class **encapsulate data and behavior** (methods) that operate on that data.

Classes vs. Objects

A key concept in **object-oriented design** is the differentiation between objects and classes. An object is any specific entity that exists in your program at run time. A class is the static thing you look at in the program listing. An object is the dynamic thing with specific values and attributes you see when you run the program. For example, you could declare a class `Person` that had attributes of name, age, gender, and so on. At run time you would have the objects *nancy*, *hank*, *diane*, *tony*, and so on—that is, specific instances of the class. If you're familiar with database terms, it's the same as the distinction between "schema" and "instance." You could think of the class as the cookie cutter and the object as the cookie.

-- Steve McConnell in *Code Complete*



Usage of Classes

A class is a collection of data and routines that share a cohesive, well-defined responsibility. A class might also be a collection of routines that provides a cohesive set of services even if no common data is involved. A key to being an effective programmer is maximizing the portion of a program that you can safely ignore while working on any one section of code. Classes are the primary tool for accomplishing that objective.

-- Steve McConnell in *Code Complete*

Classes provide a means of **bundling data and functionality together**. Creating a new class creates a new type of object, allowing new instances of that type to be made. Each class instance can have attributes attached to it for maintaining its state. Class instances can also have methods (defined by its class) for modifying its state.

-- Python documentation: <https://docs.python.org/3/tutorial/classes.html>



Classes in Python

The simplest form of class definition looks like this:

```
class ClassName:  
    <statement-1>  
    .  
    .  
    .  
    <statement-N>
```

```
class Robot(object):  
    def __init__(self, name):  
        self.name = name  
  
    def print_name(self):  
        print("My name is " + self.name)
```

```
public class Robot {  
    private String name;  
  
    public Robot(String name) {  
        this.name = name;  
    }  
  
    public void printName() {  
        System.out.println("My name is " + this.name);  
    }  
}
```

A robot class that has a name and can print out its name.

```
class Robot(object):  
    def __init__(self, name):  
        self.name = name  
  
    def print_name(self):  
        print("My name is " + self.name)
```

```
class ClassName(BaseClass):  
    def __init__(self, ...):  
        self.attribute1 = ...  
        ...  
  
    def method_1(self, ...):  
        ...
```

- `__init__` is the initialization function (constructor) to initialize/define an object of the class, can be called as: `Object = ClassName(...)` or `ClassName.__init__(Object, ...)`
- Inside the class, `self` refers to the class (the object) itself.
- dot operator “.” is used to access the attributes or methods of a class as:
`ClassName.attribute`, `ClassName.method(...)`
- (`object`) is only needed for Python 2 compatibility, can be neglected for Python 3.

Let's try it!

Inheritance

- Inheritance allows us to define a class that inherits all the attributes and methods from another class.
- Parent class is the class being inherited from, also called base class.
- Child class is the class that inherits from another class, also called derived class.
- Child class can extend/override attributes and methods of the Parent class.

```
class ParentClass(object):  
    def __init__(self, attribute1):  
        self.attribute1 = attribute1  
  
    def method_1(self, para1):  
        ...
```

```
class ChildClass(ParentClass):  
    def __init__(self, attribute1,  
                  attribute2):  
        ParentClass.__init__(self, attribute1)  
        # or use the following line:  
        # super().__init__(attribute1)  
        self.attribute2 = attribute2  
  
    def method_2(self, para2):  
        ...
```



Extend Robot to NewRobot

```
class Robot(object):  
    def __init__(self, name):  
        self.name = name  
  
    def print_name(self):  
        print("My name is " + self.name)
```



NewRobot class

- NewRobot class should inherit from the Robot class.
- NewRobot class should have a new attribute: `year_of_birth`.
- NewRobot class should have a new method: `print_age` with input `current_year`.

Let's try it!

The Four Pillars of OOP

OOP is based on the principles of abstraction, encapsulation, inheritance, and polymorphism:

- **Abstraction** allows us to focus on essential features of an object and hide irrelevant details.
- **Encapsulation** helps in keeping the data and methods together and secure from unauthorized access by external code.
- **Inheritance** allows creating new classes based on existing ones, inheriting their properties and methods.
- **Polymorphism** means that objects of different classes can be treated as if they are of the same class, making the code more flexible and reusable.

OOP provides a way to organize code in a more modular, reusable, and maintainable way. It has become the dominant programming paradigm in the software industry, and it is used in many programming languages such as Java, Python, and C++.

Source: <https://chat.openai.com/chat>





Inheritance vs Containment



Inheritance (IS-A)



"A Penguin **is a** Bird"

- **What:** Child class extends parent class
- **When to use:**
 - True hierarchical relationship
 - Need polymorphism
 - Shared core functionality
-  **Pros:** Code reuse, method overriding, Liskov substitution
-  **Cons:** Tight coupling, fragile base class problem
- **Syntax:** `class Child extends Parent` (Java), `class Child(Parent)` (Python)



Containment (HAS-A)

"A Car **has an** Engine"

- **What:** Class contains another class as member
- **When to use:**
 - Modular components
 - Dynamic behavior changes
 - Avoid inheritance hierarchies
-  **Pros:** Loose coupling, flexibility, easier testing
-  **Cons:** More boilerplate, indirect access
- **Syntax:** Member field + delegation

When to Choose?

- **Inheritance:** Strict "is-a" relationships with shared core identity
- **Containment:** "Uses-a" or "has-a" relationships, changing requirements

Source: <https://chat.deepseek.com/>



The SOLID principles

1. Single Responsibility (SRP):

"A class should have only one reason to change."

Each class/module does one thing and owns one responsibility.

2. Open/Closed (OCP)

"Open for extension, closed for modification."

Extend behavior through **inheritance/composition**, not by altering existing code.

3. Liskov Substitution (LSP)

"Subtypes must be substitutable for their base types."

Child classes **preserve parent's behavior** (no surprises when swapping implementations).

4. Interface Segregation (ISP)

"Many client-specific interfaces are better than one general-purpose one."

Avoid **fat interfaces**; split into focused, minimal contracts.

5. Dependency Inversion (DIP)

"Depend on abstractions, not concretions."

High-level modules interact via **interfaces**, not direct low-level implementations.

SOLID isn't a rulebook – it's a mindset for developing adaptable software.

Source: <https://chat.deepseek.com/>

Originally proposed in: Martin, Robert C. "Design principles and design patterns." *Object Mentor* 1.34 (2000): 597.



OOP in Python

- In Python everything is a class in Python. (try use `type()` to check)
- Guido van Rossum has designed the language according to the principle "**first-class everything**". He wrote:

"One of my goals for Python was to make it so that all objects were "first class." By this, I meant that I wanted all objects that could be named in the language (e.g., integers, strings, functions, classes, modules, methods, and so on) to have equal status. That is, they can be assigned to variables, placed in lists, stored in dictionaries, passed as arguments, and so forth."

(Blog, The History of Python, February 27, 2009)

- Everything is a class and treated the same way: functions and methods are values just like lists, integers or floats. Each of these are instances of their corresponding classes.

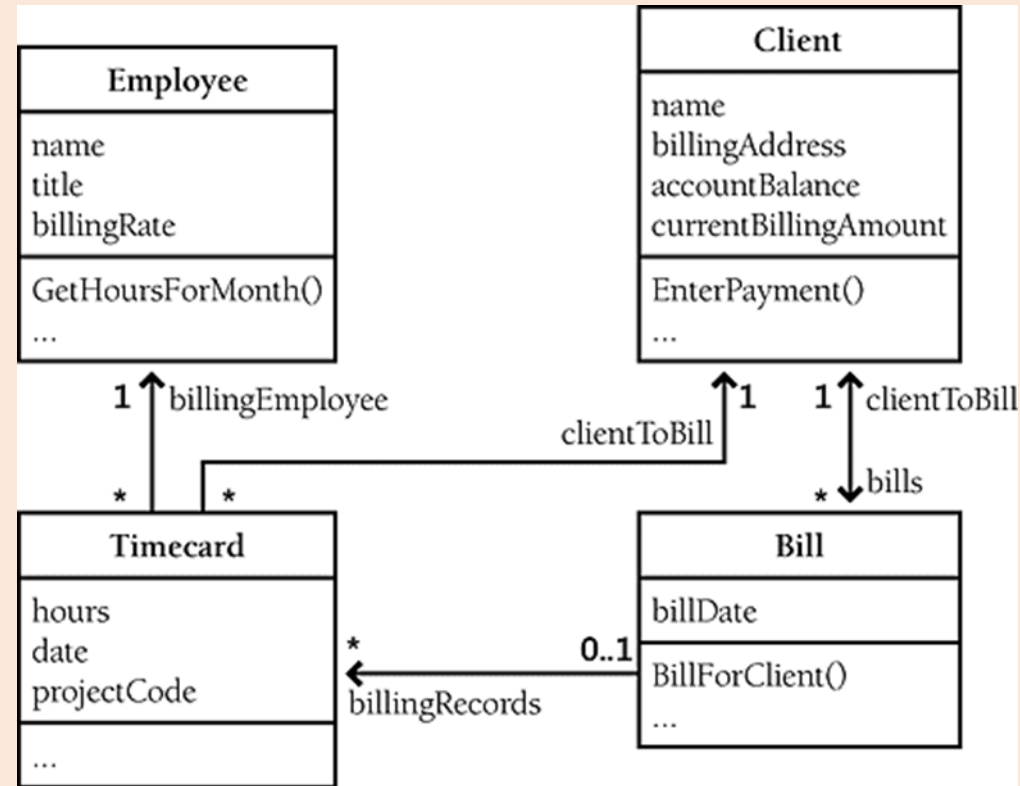
Source: <https://python-course.eu/oop/object-oriented-programming.php>



Designing with objects/classes

The steps in designing with objects are:

- Identify the objects and their attributes (methods and data).
- Determine what can be done to each object.
- Determine what each object is allowed to do to other objects.
- Determine the parts of each object that will be visible to other objects—which parts will be public and which will be private.
- Define each object's public interface.



A billing system composed of four major objects.

Source: McConnell, Steve. Code complete. Pearson Education, 2004.

Questions?



*Remember, you're expected to work
about 6 hours outside of class. Schedule
accordingly.*

Homework.

- **Homework 0:** Write your own Turbine classes, do it individually.
- **Homework 1:** Write a class for your final project, do it with your group.
- **Homework optional:** Transforming your CodeCamp project with OOP.
- We'll open BORs in a minute. Enter room corresponding to your Team ID (on Learn).
- **To get help during class:** Post in Slack / #debugging, if you want a TA to enter your BOR or come find your group.

Any questions?



References

- McConnell, Steve. Code complete. Pearson Education, 2004.
- Ario Liyan, (2023) What is a programming paradigm? [link: <https://medium.com/@Ariobarxan/what-is-a-programming-paradigm-ec6c5879952b>]
- <https://docs.python.org/3/tutorial/classes.html>
- <https://python-course.eu/oop/object-oriented-programming.php>
- Python Metaclasses: Everything is an Object [link: <https://www.youtube.com/watch?v=uLPnBaUhjKU>]
- Martin, Robert C. "Design principles and design patterns." Object Mentor 1.34 (2000): 597.
- <https://en.wikipedia.org/wiki/SOLID>
- SOLID Principles: Do You Really Understand Them? [link: <https://www.youtube.com/watch?v=kF7rQmSRlq0>]

