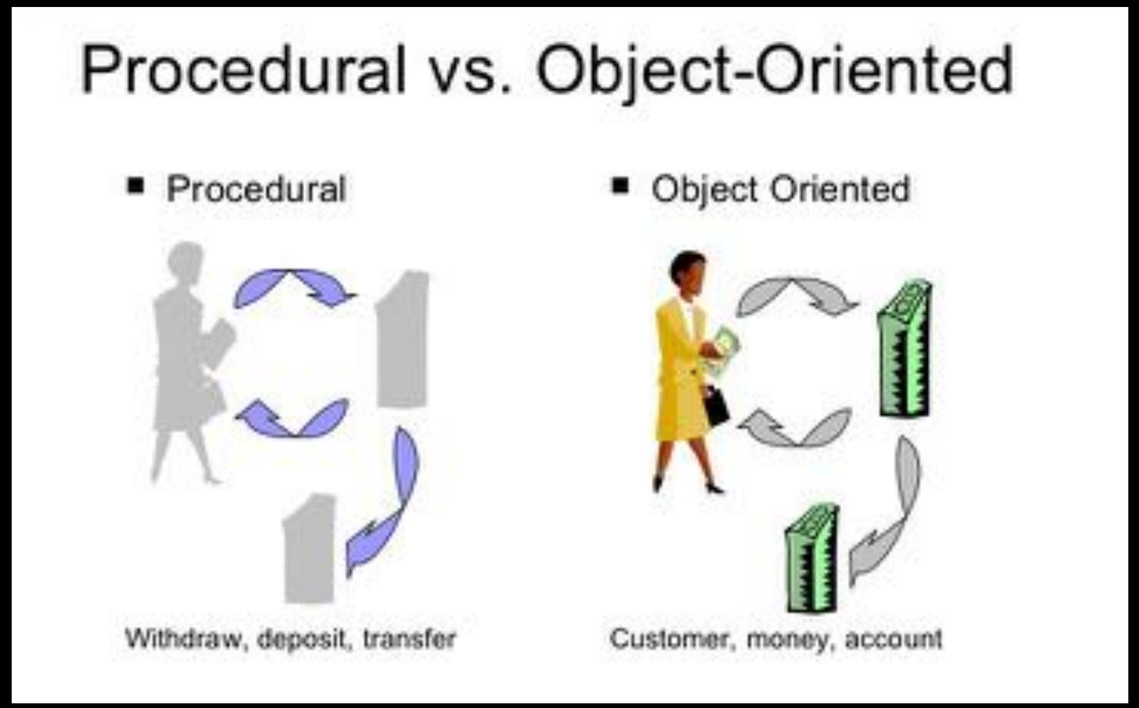
Object-Oriented Programming and other modern coding concepts in Python



Cameron Hummels
NSF Fellow, Caltech

Code should* be:

Code should* be:

- **correct**
- **efficient**
- **simple**
- **easy to read**
- **concise**
- **accessible**

Code should* be:

- correct → Tests
- efficient
- simple → OOP
- easy to read
- concise
- accessible → Repository

What is object-oriented programming?

A programming paradigm based on the abstraction of objects to represent a real-world environment enabling simpler, modular, and less-redundant code

Three primary types of programming:

procedura

- top-down
- group data as procedures
- identify tasks
- C, Fortran, Basic

object-oriented

- bottom-up
- group data as objects
- identify objects
- C++, Java, Python

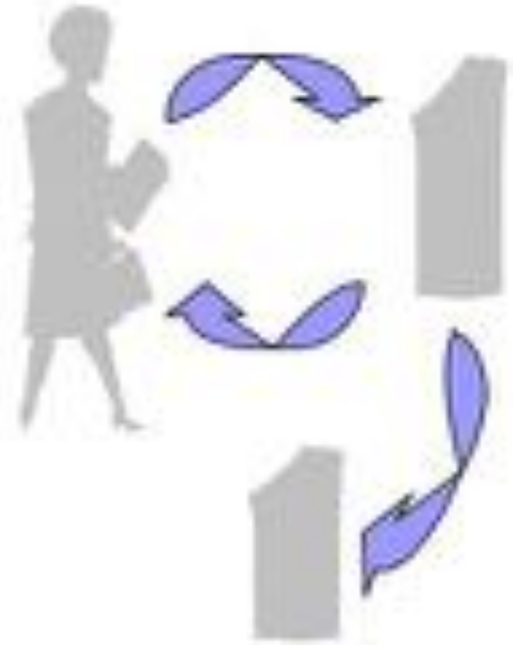
functional

- recursive
- groups data as functions
- LISP, scheme

Three primary types of programming:

Procedural vs. Object-Oriented

- Procedural



Withdraw, deposit, transfer

- Object Oriented

Customer, money, account

Traditional procedural programming

```
task 1;
task 2;
task 3;
   ...
```

Traditional procedural programming

```
def procedure/function1():

def prodedure/function2():

main:
    function1()
    function2()
```

Object-oriented programming

```
class class1:
    attributes
    methods

main:
    objectA = class1()
    objectA.method()
```

Object-oriented programming concepts

- encapsulation (data hiding)
- data abstraction (black box)
- inheritance (subclasses)
- polymorphism (overloading functions)
- variable scope (enclosed namespaces)

Object-oriented programming concepts

- encapsulation (data hiding)
  - public view of objects; constrained interaction
- data abstraction (black box)
- inheritance (subclasses)
- polymorphism (overloading functions)
- variable scope (enclosed namespaces)

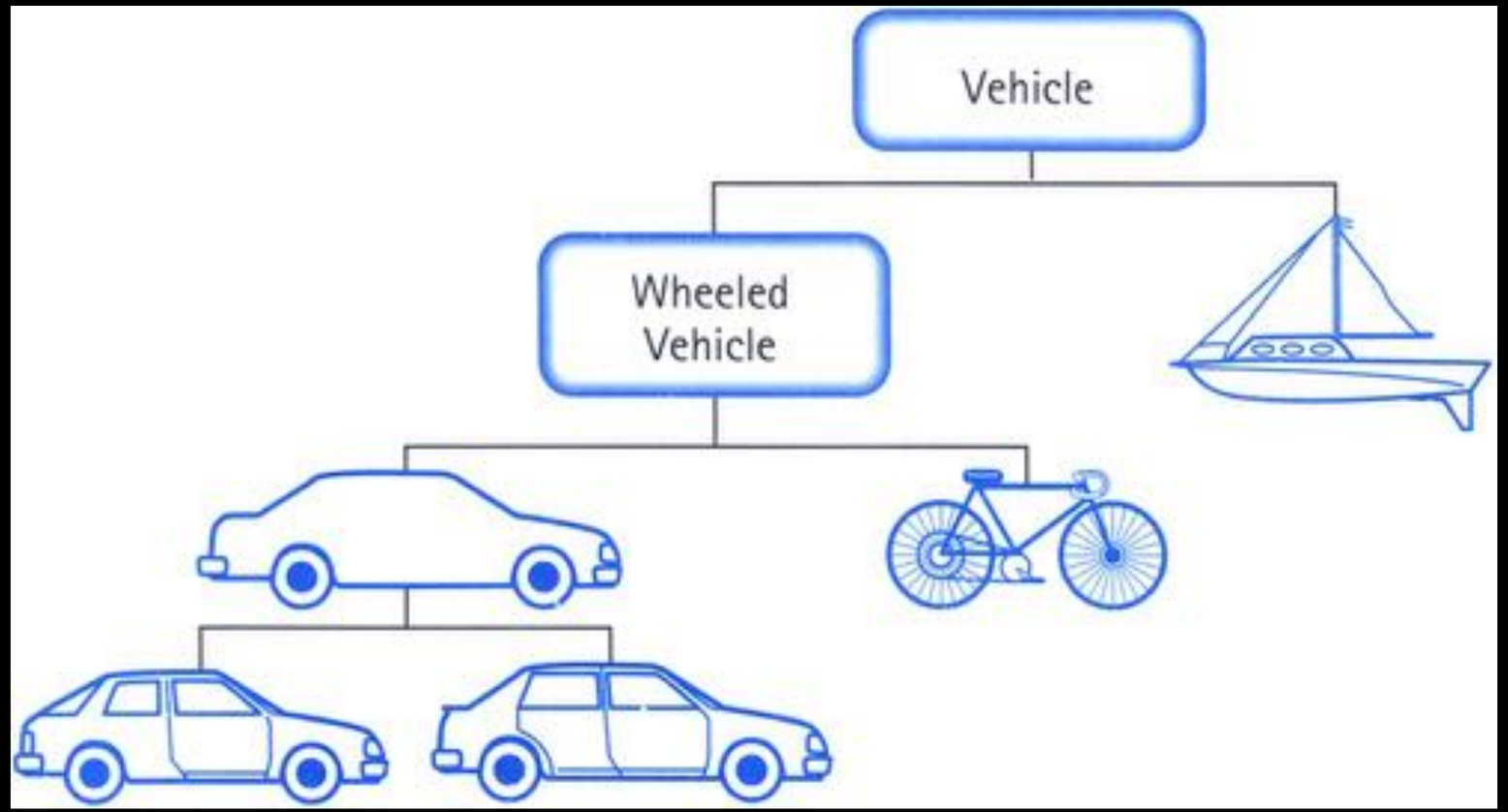
Object-oriented programming concepts

- encapsulation (data hiding)
- data abstraction (black box)
  - API: Application Programming Interface
- inheritance (subclasses)
- polymorphism (overloading functions)
- variable scope (enclosed namespaces)

Object-oriented programming concepts

- encapsulation (data hiding)
- data abstraction (black box)
- inheritance (subclasses)
  - modular code; avoids redundancy
- polymorphism (overloading functions)
- variable scope (enclosed namespaces)

Object-oriented programming concepts



Object-oriented programming concepts

- encapsulation (data hiding)
- data abstraction (black box)
- inheritance (subclasses)
- polymorphism (overloading functions)
  - one function name can do many things
- variable scope (enclosed namespaces)

Object-oriented programming concepts

- encapsulation (data hiding)
- data abstraction (black box)
- inheritance (subclasses)
- polymorphism (overloading functions)
- variable scope (enclosed namespaces)
  - variables can be different in different contexts

Live-coding Example

Concepts to remember

- encapsulation (data hiding)
- data abstraction (black box)
- inheritance (subclasses)
- polymorphism (overloading functions)
- variable scope (enclosed namespaces)

Concepts to remember

- within a class:
  - function -> method
  - variable -> attribute
- self and super
- attributes can be private!
- CamelCase -> class names
- snake_case -> method, variable names
- be descriptive in variable naming
- docstrings and comments!

Concepts to remember

- if __name__ == '__main__': for body of program
- classes should be initialized with def __init__()
- include a __repr__() method so can print
- the dir(), and help() commands provide context!

Code design concepts

- think about goals of code
- break into reasonable classes
- pseudocode it up
- check for efficient algorithm*
- test each function and class!
- assemble code in __main__
- run
- optimize if necessary