# Object-Oriented Programming, Python, and HARK

David Low (CFPB)

April 26, 2017

# What Is Object-Oriented Programming (OOP)?

Traditional programming is *procedure*-oriented.

- Basically, do things as you think of them.
- Focus is on the *procedure* necessary to accomplish goals.
- Most immediately intuitive way to program (or do anything.)

Procedure-oriented pseudo-code to grow tomatoes and strawberries:

1. Plant strawberry seed
2. Water strawberry
3. Pick strawberries in June
4. Plant tomato seed
5. Water tomato
6. Pick tomatoes in August

# What Is Object-Oriented Programming?

Object-oriented programming is (ahem) *object*-oriented.

- Focus on *objects* (abstract structures) necessary to accomplish goals.
- Less immediately intuitive... but often far more useful.

It is useful in the same way abstraction is useful in any other field

- History: what do revolutions have in common?
  - Weak government, fractured elite, don't start off as revolutions, etc.
- Math: what do metric spaces have in common?
  - Defined by a metric, which in turn defines open and closed sets
- Programming: what should chunks of my code have in common?
  - Helps clarify thinking, like abstraction in other fields
  - Benefits unique to programming: avoid repetition; cleaner code

# What Is Object-Oriented Programming?

In our example, tomatoes and strawberries are both fruit plants.

Object-oriented pseudo-code:

1. tomato = FruitPlant(harvest_time = August)
2. tomato.plantSeed()
3. tomato.water()
4. tomato.pickFruit()
5. strawberry = FruitPlant(harvest_time = June)
6. strawberry.plantSeed()
7. strawberry.water()
8. strawberry.pickFruit()

Fruit plants "know" how (and when!) to plant, water, and pick themselves.

# Why Object-Oriented Programming?

Clarifies your thinking

- Recognizing similarities and differences between tomatoes and strawberries might help you think about blueberries or Redwood trees or Venus fly traps, too.
- Higher-level thinking is much more fun!

Lets you avoid repetition, which is hugely important in programming

- Less effort
- Cleaner code
- Easier to debug

These things are nice for small projects. They are *amazing* for big projects.

Still, this might seem abstract – it did to me at first.
Becomes more obvious with experience.

# Why Python? (This one's easy!)

Main reason to learn Python over any other comparably high-level scientific computing language (Matlab; Julia) is OOP

Python provides amazing support for OOP (though it also allows procedure-oriented programming)

Python makes it extremely easy to write good, clean, clear, reusable code

- These qualities are especially important for large projects (e.g. code for your job market paper, or HARK)

# Basic OOP (in Python)

A *class* is an abstract grouping, members of which have stuff in common

- E.g., a fruit plant

An *instance* is a specific implementation of a class

- E.g., a tomato or strawberry

An *attribute* is just an object attached to a class instance.

- Usually, but not always, instances of a class have the same attributes
- E.g. whether or not the plant is watered

A *method* is a function attached to an instance that also operates on it

- This sounds complex. Really, just how instances "know" how to do important things to themselves. E.g. tomato.pickFruit()

# Inheritance

How should objects be grouped together into classes? Often, they will have some things in common, but not others.

Good way to deal with this is called *inheritance*:

- Broad classes; instances share a few things in common (e.g. plants)
- Narrow classes; instances have more in common (e.g. fruit plants)
- Classes can *inherit* from other classes; avoid redefining attributes
  - So narrow classes can *inherit* from broader ones
  - E.g. fruit plants are plants

Classes can inherit from multiple other classes

- Called *multiple inheritance*
- Fruit trees are fruit plants, but they are also things to climb.

# Code Exploration

# Now on to HARK!

[But first, a 10 minute break.]

# OOP and HARK

Whole idea behind HARK: stop solving models from scratch

- If the model has been solved before, use that code
- If the model hasn't been solved before, take code for a similar model, and modify it only where necessary

This is conceptually almost identical to OOP

How? HARK thinks of types of agents as classes

- Matt White will give a lot more detail when he visits
- Agents in HARK "know" how to solve their model, through the *solve*() method
- If desired, they can then interact with other agents with potentially different parameters or problems

# HARK and OOP 1

Example 1: Import the model, set a few parameters, and solve it

E.g. MPC out of credit vs MPC out of temporary income

Currently two other demos in HARK:

1. Precautionary saving and the Great Recession
2. Precautionary saving and Chinese growth

# HARK and OOP 2

Example 2: Import one model, change it a little bit, and solve new model.

HARK includes the "standard" consumption-savings model.
It also includes two extensions:

1. $R^{debt} \neq R^{savings}$
2. Multiplicative utility shocks

What about both at the same time?

KinkyPrefConsumerType in ConsPrefShockModel.py, combination of:

1. KinkedRConsumerType from ConsIndShockModel.py
2. PrefShockConsumerType, from ConsPrefShockModel.py

Imagine doing this without OOP!

# Conclusion

OOP, Python, and HARK are all great.

You should learn them!