OFFICE OF FINANCIAL RESEARCH

Software Architecture of ARK

Christopher D. Carroll[1], Alexander M. Kaufman[2],
David C. Low[2], Nathan M. Palmer[3], Matthew N. White[4]

1. Johns Hopkins University
2. Consumer Financial Protection Bureau
3. Office of Financial Research
4. Department of Economics, University of Delaware

Federal Reserve Board, September 2016

Language of ARK: Python

Why Python?

Why Python?

- ("Why not Julia?")

Why Python?

- ("Why not Julia?")
- Quick answer: community support, libraries, and object orientation*

Python Community Support

Python routinely ranked as a top language: by projects, discussions, job openings, searches for tutorials

Python Community Support

Python routinely ranked as a top language: by projects, discussions, job openings, searches for tutorials

General use: ranked top 2-5

- PYPL, GitHut, RedMonk, TIOBE*

Python Community Support

Python routinely ranked as a top language: by projects, discussions, job openings, searches for tutorials

General use: ranked top 2-5

- PYPL, GitHut, RedMonk, TIOBE*

Special-purpose:

- Engineering: IEEE Spectrum, top 2-3 (C, Java)

Python Community Support

Python routinely ranked as a top language: by projects, discussions, job openings, searches for tutorials

General use: ranked top 2-5

- PYPL, GitHut, RedMonk, TIOBE*

Special-purpose:

- Engineering: IEEE Spectrum, top 2-3 (C, Java)
- Data science: O'Reilly Data Science Survey "top 4 tools"

Python Community Support

Python routinely ranked as a top language: by projects, discussions, job openings, searches for tutorials

General use: ranked top 2-5

- PYPL, GitHut, RedMonk, TIOBE*

Special-purpose:

- Engineering: IEEE Spectrum, top 2-3 (C, Java)
- Data science: O'Reilly Data Science Survey "top 4 tools"
  - SQL, Excel

Python Community Support

Python routinely ranked as a top language: by projects, discussions, job openings, searches for tutorials

General use: ranked top 2-5

- PYPL, GitHut, RedMonk, TIOBE*

Special-purpose:

- Engineering: IEEE Spectrum, top 2-3 (C, Java)
- Data science: O'Reilly Data Science Survey "top 4 tools"
  - SQL, Excel
  - R, Python

Python Community Support

Python routinely ranked as a top language: by projects, discussions, job openings, searches for tutorials

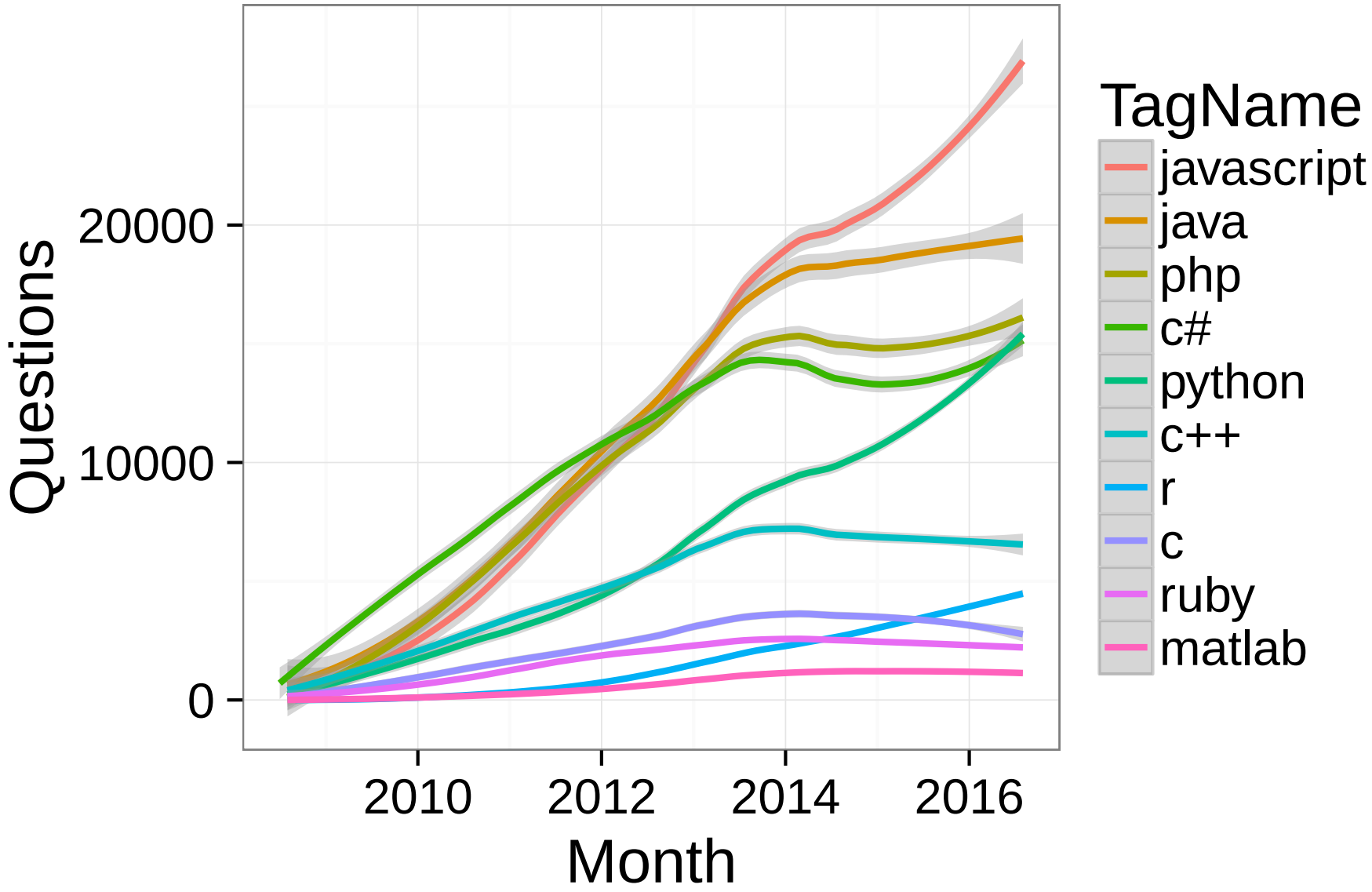
General use: ranked top 2-5

- PYPL, GitHut, RedMonk, TIOBE*

Special-purpose:

- Engineering: IEEE Spectrum, top 2-3 (C, Java)
- Data science: O'Reilly Data Science Survey "top 4 tools"
  - SQL, Excel
  - R, Python
- Statistics, via CrossValidated tags: 3rd (R, SPSS)

StackOverflow Question Tags



Organizations and Libraries

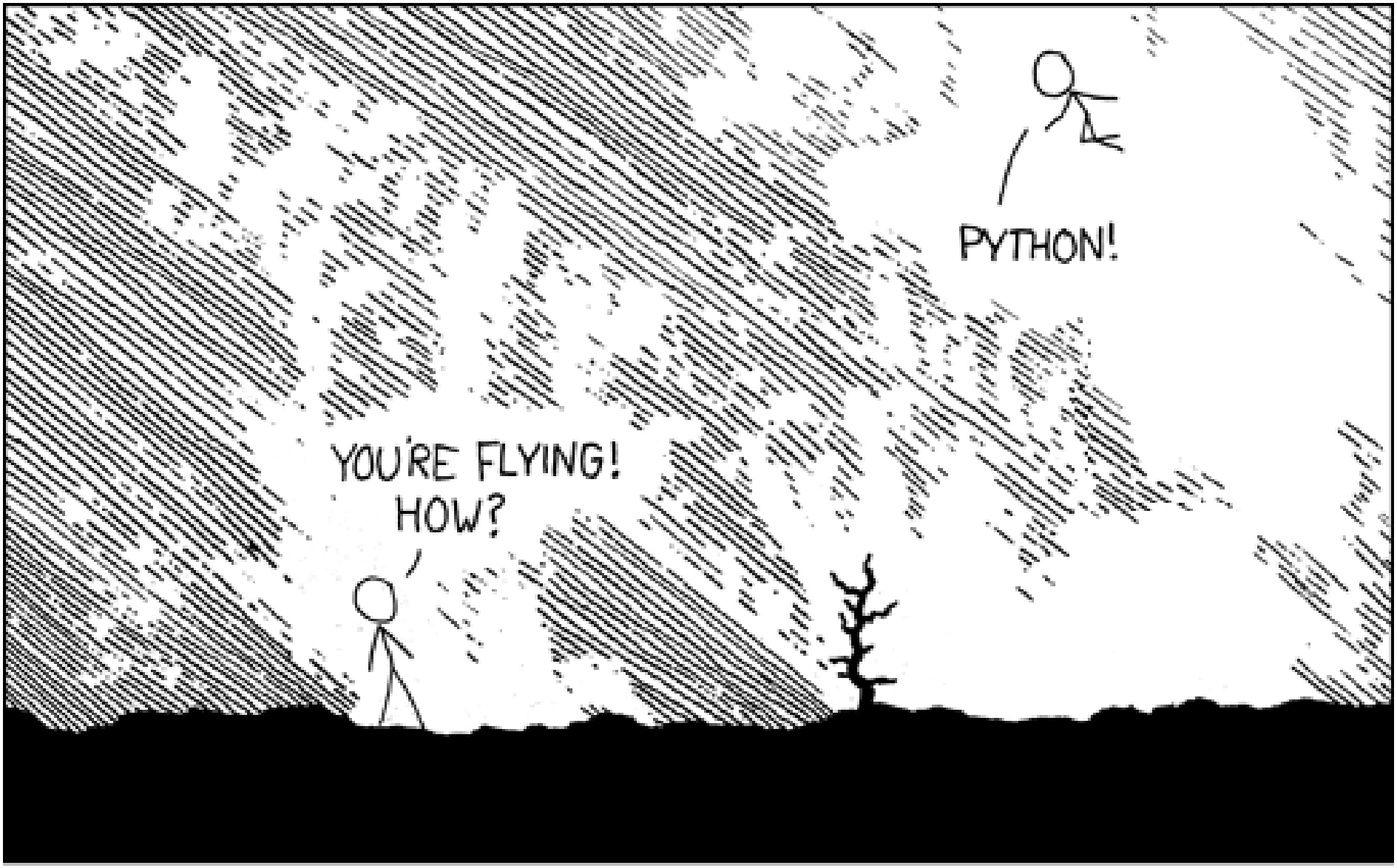
Organizations supporting:

- NASA, Bloomberg, IBM, Los Alamos, Lawrence Livermore, many more

Libraries:

- Anaconda (package system)
- NumPy, SciPy, Pandas, Statsmodels, Scikit-learn, Numba, Spark
- Quant-Econ, NetworkX, AstroPy, PyMC3
- Too many to list:
  - compilers, grid and multiprocessing computing, GPU support, web scraping, NLP
- nflgame ...

import antigravity



Some Anecdotes

"Why Python is steadily eating other languages' lunch" goo.gl/OUZet9

- Computational neuro: "3 years ago, five languages; now Python"

Some Anecdotes

"Why Python is steadily eating other languages' lunch" goo.gl/OUZet9

- Computational neuro: "3 years ago, five languages; now Python"

John Cochrane, "Eight young stars:"

> *If you're going in to economics these days, learn python, R, stata, html, java; know how to scrape data from the web, run a large programming task in a disciplined style, manipulate and clean large data sets. That's the key intellectual arbitrage behind the young stars' work today, and way more important than measure theory and real analysis!*

But What about Speed?

Important speed notes:

- Programmer time binds before execution time
- When execution time binds, many paths forward
  - compiled code (eg. python-compilers-workshop.github.io)
  - multiprocessing
  - grid computing
- Extensive libraries to address these

Aruoba & Fernández-Villaverde, JEDC (2015)

Table 1: Average and Relative Run Time (Seconds)

	Mac			Windows		
Language	Version/Compiler	Time	Rel. Time	Version/Compiler	Time	Rel. Time
C++	GCC-4.9.0	0.73	1.00	Visual C++ 2010	0.76	1.00
Fortran	GCC-4.9.0	0.76	1.05	GCC-4.8.1	1.73	2.29
Java	JDK8u5	1.95	2.69	JDK8u5	1.59	2.10
Julia	0.2.1	1.92	2.64	0.2.1	2.04	2.70
Matlab	2014a	7.91	10.88	2014a	6.74	8.92
Python	Pypy 2.2.1	31.90	43.86	Pypy 2.2.1	34.14	45.16
	CPython 2.7.6	195.87	269.31	CPython 2.7.4	117.40	155.31
R	3.1.1, compiled	204.34	280.90	3.1.1, compiled	184.16	243.63
	3.1.1, script	345.55	475.10	3.1.1, script	371.40	491.33
Mathematica	9.0, base	588.57	809.22	9.0, base	473.34	626.19
Matlab, Mex	2014a	1.19	1.64	2014a	0.98	1.29
Rcpp	3.1.1	2.66	3.66	3.1.1	4.09	5.41
Python	Numba 0.13	1.18	1.62	Numba 0.13	1.19	1.57
	Cython	1.03	1.41	Cython	1.88	2.49
Mathematica	9.0, idiomatic	1.67	2.29	9.0, idiomatic	2.22	2.93

Object Oriented Programming in ARK

Object Oriented Programming

OO programming:

- Define your own complicated variable types
- . . . with instance-specific values and functionality

Object Oriented Programming

OO programming:

- Define your own complicated variable types
- . . . with instance-specific values and functionality

Example: a "regression object"

- attributes: data, coefficients, errors, p-vals
- methods (functions): minimize errors, calculate variance, forecast, plot

Extend: calculate variance differently

Usefulness of Object Orientation

- Modular code: define basic structure required for a solution
  - All code that matches can run under common framework
  - De facto "API"

Usefulness of Object Orientation

- Modular code: define basic structure required for a solution
  - All code that matches can run under common framework
  - De facto "API"
- Inheritance (reduce code replication)
- De facto heterogeneity in objects

Modular structure of a MicroDSOP

Microeconomic solution:

- Define terminal (initial) value, policy functions
- Step back one period, solve for T-1 value, policy functions
- Repeat process until t=0 or convergence criteria met for value, policy

Modular structure of a MacroDSOP

Macroeconomic KS-style solution:

- Define initial beliefs about aggregate dynamics rule
- Create simulation history:
  - Provide agents state variables
  - Agents solve problems, markets clear, step forward, repeat
- Generate new beliefs from simulation history
- Repeat until attain convergence criteria

Modular Microeconomic Structure

Micro base class: `AgentType`

- General purpose class for representing economic agents
- Each model creates a subclass of `AgentType`
  - Includes model-specific attributes, functions, and methods...
  - ...And how to solve the "one period problem" for that model
  - Instances of subclass are *ex ante* heterogeneous "types"

Modular Microeconomic Structure

Micro base class: `AgentType`

- General purpose class for representing economic agents
- Each model creates a subclass of `AgentType`
  - Includes model-specific attributes, functions, and methods...
  - ...And how to solve the "one period problem" for that model
  - Instances of subclass are *ex ante* heterogeneous "types"
- All `AgentType` subclasses use the same `solve()` method
- Just a universal backward induction loop...
- ...That lets different models "play nicely" together

Modular Macroeconomic Structure

Macro base class: `Market`

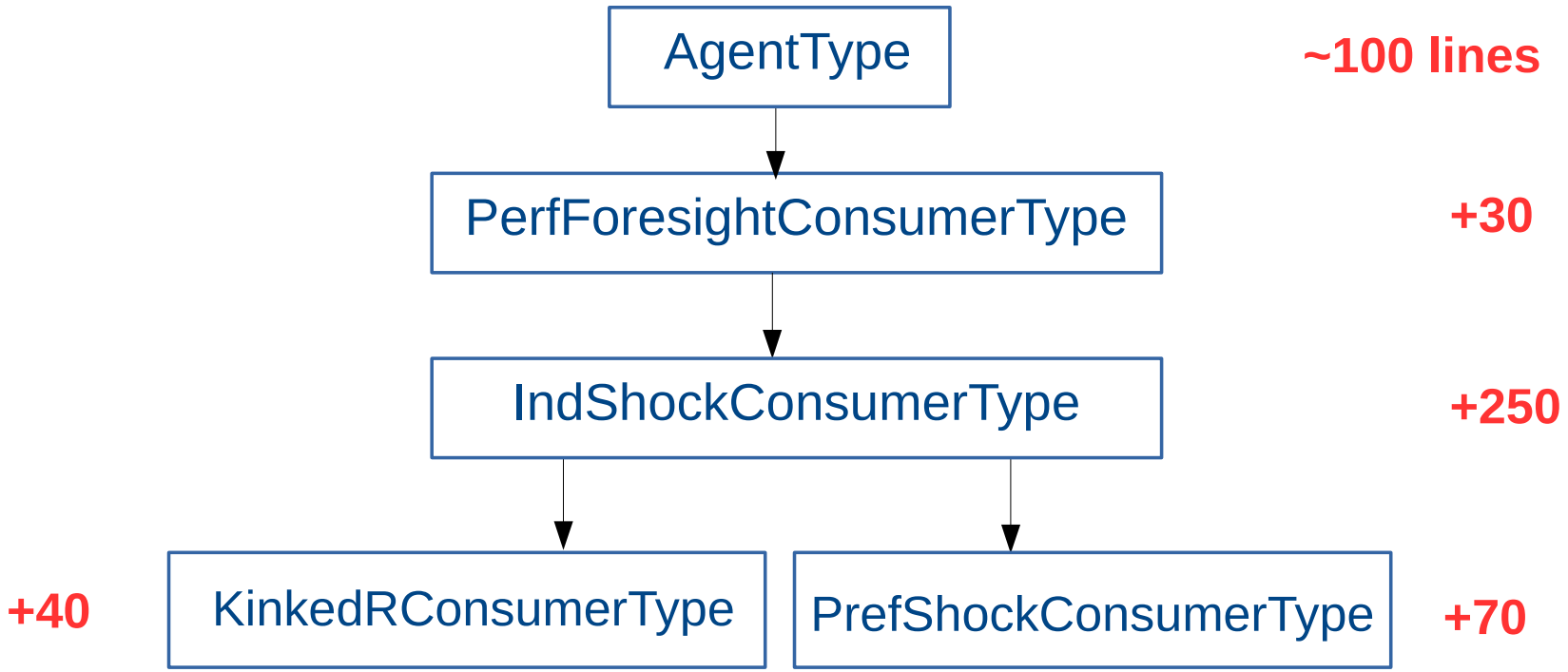
- Instance represents an "outcome aggregator"
- Turns microeconomic outcomes into macroeconomic outcomes
  - E.g. asset holdings of agents –> aggregate capital –> R and w

Modular Macroeconomic Structure

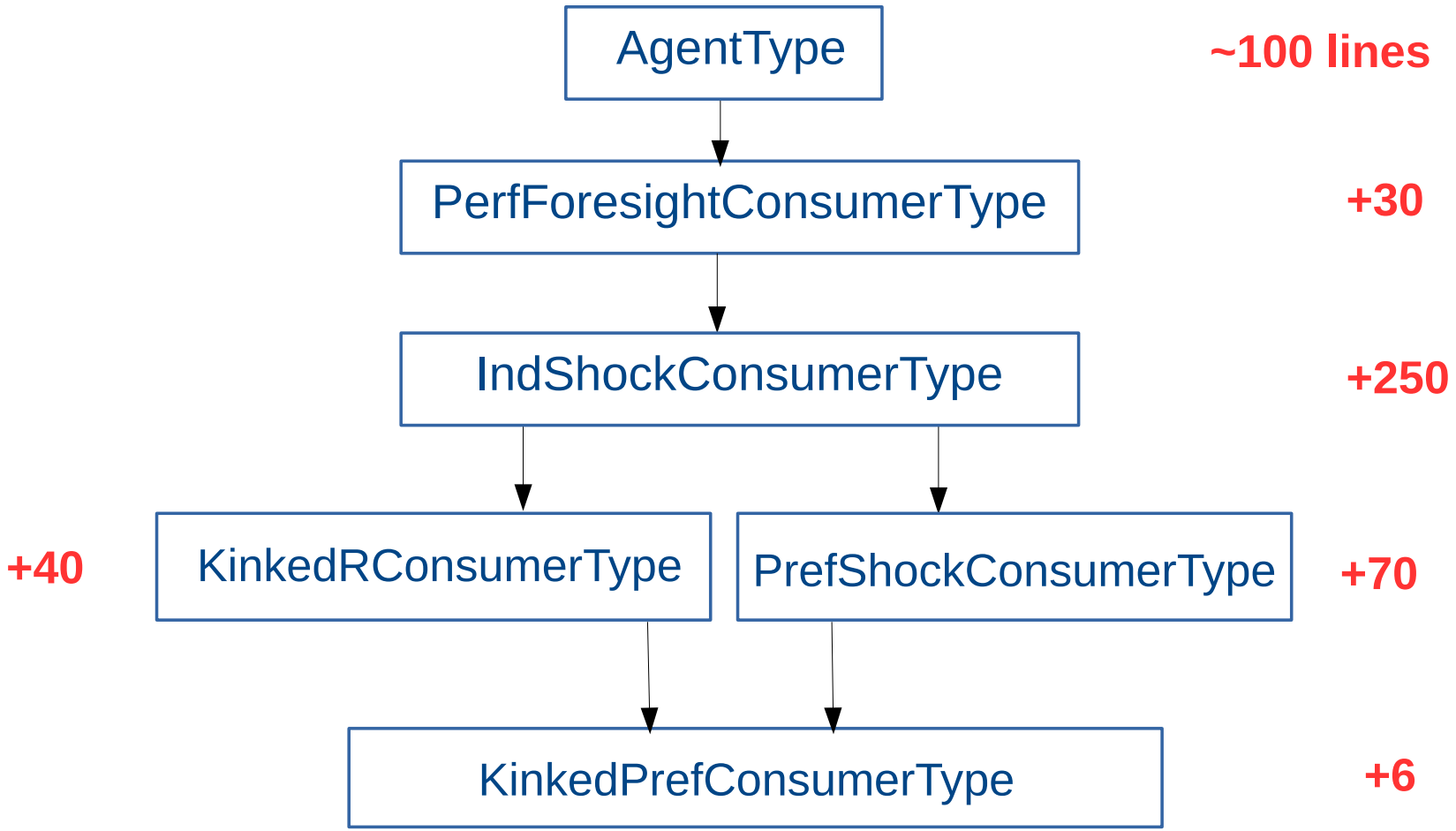
Macro base class: `Market`

- Instance represents an "outcome aggregator"
- Turns microeconomic outcomes into macroeconomic outcomes
  - E.g. asset holdings of agents –> aggregate capital –> R and w
- Also has "universal solver" to find general equilibrium
  - Seeks consistency in agent beliefs about macro outcomes...
  - ...and the macro outcomes that occur when they act on beliefs

Simple Class Inheritance Diagram



Simple Class Inheritance Diagram



Additional Tools from Software Development

Technology Improvement: Efficient Code Output

- Automated Documentation
  - Inline
  - Online (Sphinx)
- Automated Testing
  - Unit testing
  - Peer review
- Version Control
  - Automatically archive code
  - Collaborative workflows (tracking issues, decentralized review)
  - "Freeze" scientific publications for reference
- Scientific Notebooks
  - Combine code, math, descriptions, interaction ("vignettes")

Automated Documentation

Types of documentation:

- system-style
- on-line (API)

Automated Documentation

```
def utility(c, gamma):
    """
    Return constant relative risk aversion (CRRA) utility of consumption "c"
    given risk aversion parameter "gamma."

    Parameters
    ----------
    c: float
        Consumption value.
    gamma: float
        Risk aversion, gamma != 1.

    Returns
    -------
    u: float
        Utility.

    Notes
    -----
    gamma cannot equal 1. This constitutes natural log utility; np.log
    should be used instead.
    """
    u = c**(1.0 - gamma) / (1.0 - gamma)  # Find the utility value of c given gamma
    return u                              # Return the utility value
```

Automated Documentation

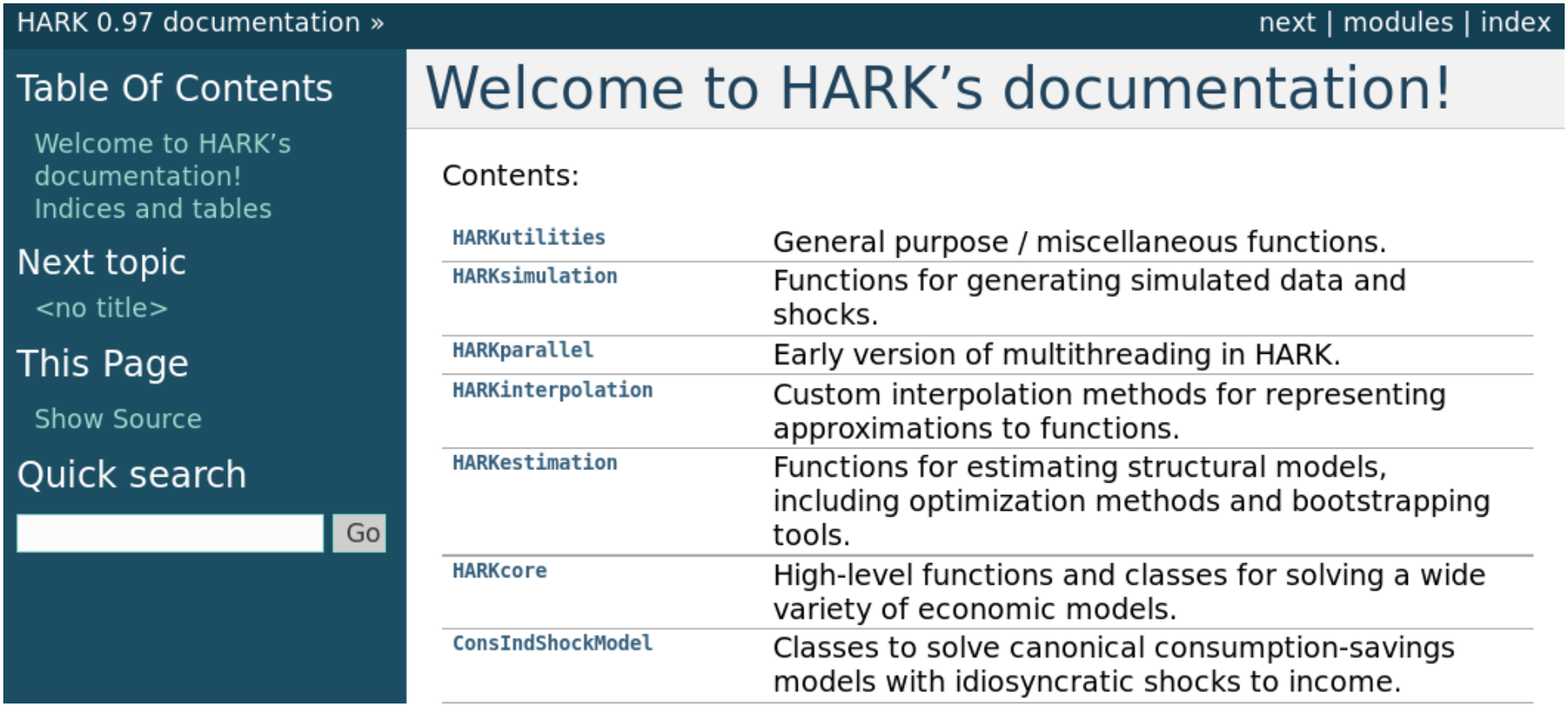


Figure : Sphinx Docs

Testing

Unit tests: small tests at simple functional levels

- doctests
- unitttest
- Automatically run tests, examine test coverage
- Broader acceptance protocols: determining when a piece of code is acceptable

Doctest Example

```
def utility(c, gamma):
    """
    Return CRRA utility of consumption "c" given risk aversion parameter "gamma."

    ...(excluded for brevity)...

    Tests
    -----
    Test a value which should pass:
    >>> utility(1.0, 2.0)
    -1.0

    Test a value which should fail:
    >>> utility(1.0, 1.0)
    Traceback (most recent call last):
        ...
    ZeroDivisionError: float division by zero
    """
    return( c**(1.0 - gamma) / (1.0 - gamma) )
```

Figure : Doctest Example

Version Control

Two parts: **version control** system (eg. Git) and **repository hosting** system (eg. Github, Stash, Bitbucket, etc. . .)

- Git: basic command-line system for managing archiving and workflow

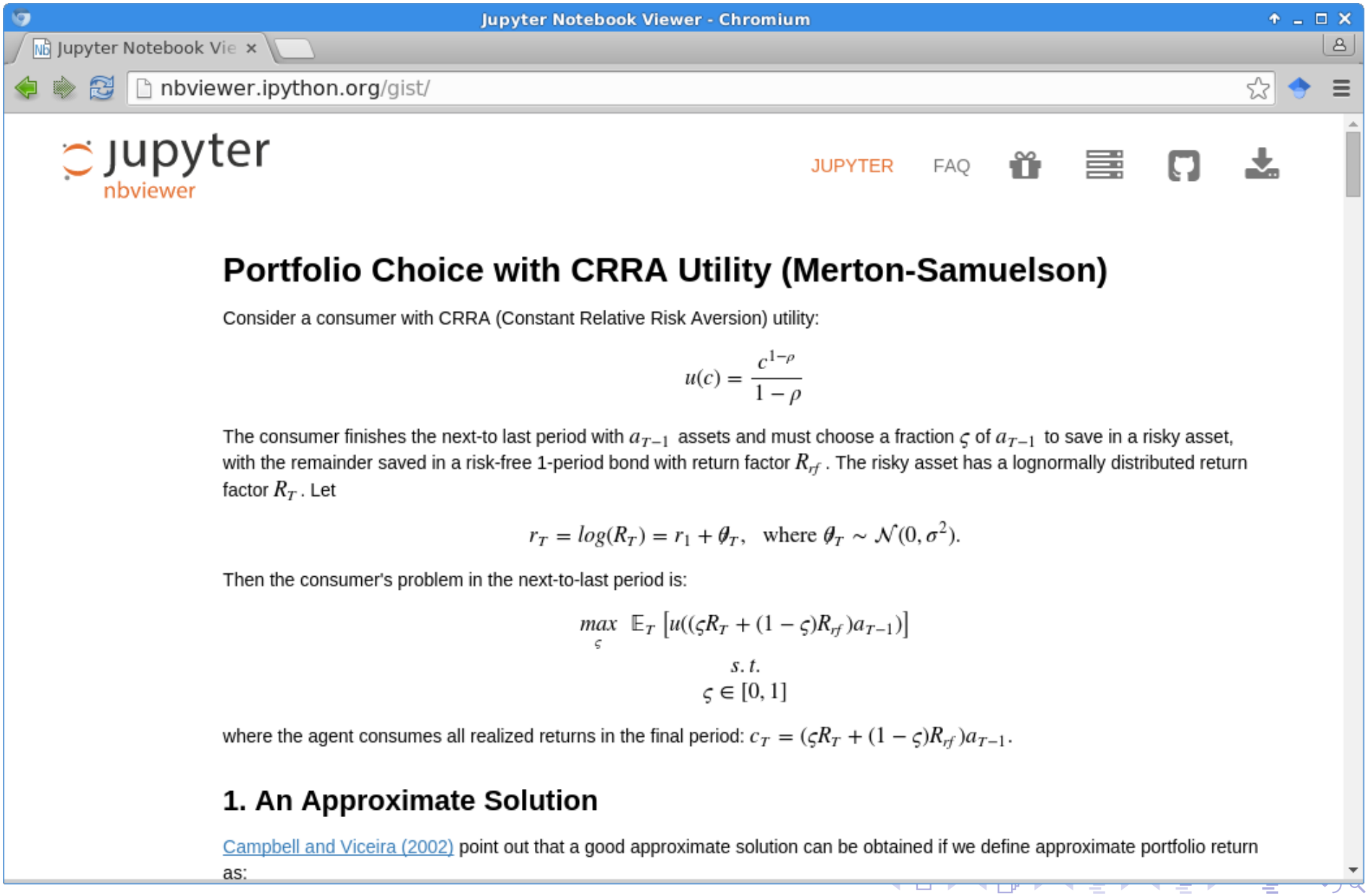
```
localhost:~/workspace/solvingmicrodsop$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   Python/Exploring-Estimation-Module.ipynb
        modified:   Python/Histogram_of_Beta_Bootstrap_Sample.png
        modified:   Python/Histogram_of_Rho_Bootstrap_Sample.png
        modified:   Python/params_json_replication.json

no changes added to commit (use "git add" and/or "git commit -a")
localhost:~/workspace/solvingmicrodsop$
```

- Github/Stash/Bitbucket/etc: centralized repository hosting with web interface

Scientific Notebooks: Jupyter

- Simply reviewing scientific code can take time
- Solution: notebooks combining code, math, visualization, interaction
- Must be:
  - low cost for researcher end-user
  - easy to use, share
  - archivable

Scientific Notebooks: Jupyter



Bring It Together

Example of research code "vignettes:"

- Create Jupyter notebook
- Host on Github/Bitbucket/etc as Gist + scientific archive
- Post to NBViewer
- With unit testing, easy to review

Next steps: Specifics of ARK

Questions?