Things I Wish I Knew When I First Started Learning Python for Data Stuff

A Highly Opinionated Guide

N. Dorukhan Sergin | February 27 2021 | IISE Western Regional Conference

A Typical Industrial Engineering Role

JOB REQUIREMENTS

- Currently pursuing an undergraduate degree in Industrial Engineering or similar field
- Proficient in use of Microsoft Office software, including Access, Excel, Word and PowerPoint

The Future is IE + Coding

Data Scientist - Operations

at Stitch Fix (View all jobs)

SAN FRANCISCO, CA

ABOUT THE TEAM

In this role you will use Operations Research, Data Science methods and tools to plan and optimize operations at Stitch Fix. You will build algorithmic solutions to solve business problems around demand and capacity planning, personalized allocation, logistics and warehouse operations - all with the goal to support client satisfaction and company growth.

WE'RE EXCITED ABOUT YOU BECAUSE YOU HAVE...

- 2+ years of experience supporting operational roles (demand planning, supply planning, capacity management, operations)
- A Ph.D. or Masters degree in Statistics, Biostatistics, Marketing, Econometrics, Mathematics, or other quantitative fields
- Experience writing code (Python preferred) in production environments where you have learned industry practices such as unit testing and code reviews.
- The ability to quickly and iteratively prototype analyses and algorithms
- A deep understanding business processes, and have experience working with people with different backgrounds, priorities, and responsibilities
- The ability to explain complex concepts well and move decisions forward in collaboration with your business partners
- An inquisitive nature, and you scrutinize functional efforts through the lens of constantly improving the client experience and the business

The Future is IE + Coding

Data Scientist, Charging Data & Modeling

Job Category	Engineering & Information Technology
Location	Palo Alto, California
Req. ID	83177
Job Type	Full-time

Requirements

- Bachelor's, Master's or PhD in a related field (e.g., CS, Operations Research, Software Engineering, Statistics)
- Strong programming skills with a solid foundation in data structures and algorithms
- Proficiency in data analysis, modeling, and web services in Python
- Proficiency in SQL relational databases and/or NoSQL databases
- Experience with statistical data analysis and machine learning such as linear models, time-series forecasting, or neural networks
- Smart but humble, with a bias for action


THEY TOOK
ER JERBS!!!

Who Am I?

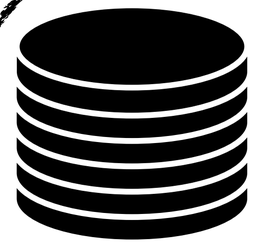
- 4 years into PhD in IE
  - **Research Focus:** Deep Learning in Manufacturing
- B.S. and M.S. also in IE
- +7 years of using Python
  - **As a Researcher:** Run Experiments / Plot Charts
  - **As a Machine Learning Engineer Intern:** Deploy Machine Learning Software
  - **As a Risk Analytics Intern:** Automate Monthly Report Generation
  - **As a Hobbyist:** Develop websites, hobby projects

Who I Think You Are?

Things I Wish I Knew When I First Started Learning **Python for Data Stuff**

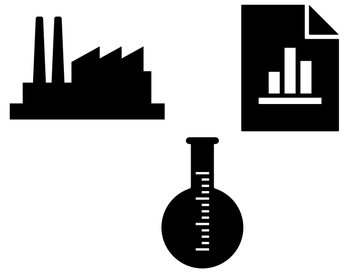
Industrial/Systems Engineer

Have no formal software development experience



Turn Data

Using Python



Data-driven Decision Making Products / Research Papers

This Guide is…

- for people who want to do serious engineering
- for people who want to deliver production-ready Python-based solutions
- not an exhaustive list of all the things needs to be learned
- highly opinionated
- not to be remember but to be understood

Case Study: Simulating the M/M/1 Queue

Task: Plot the histogram of wait times (for a given lambda, mu and a simulation run-length)

Image Source: https://en.wikipedia.org/wiki/M/M/1_queue#/media/File:Mm1_queue.svg

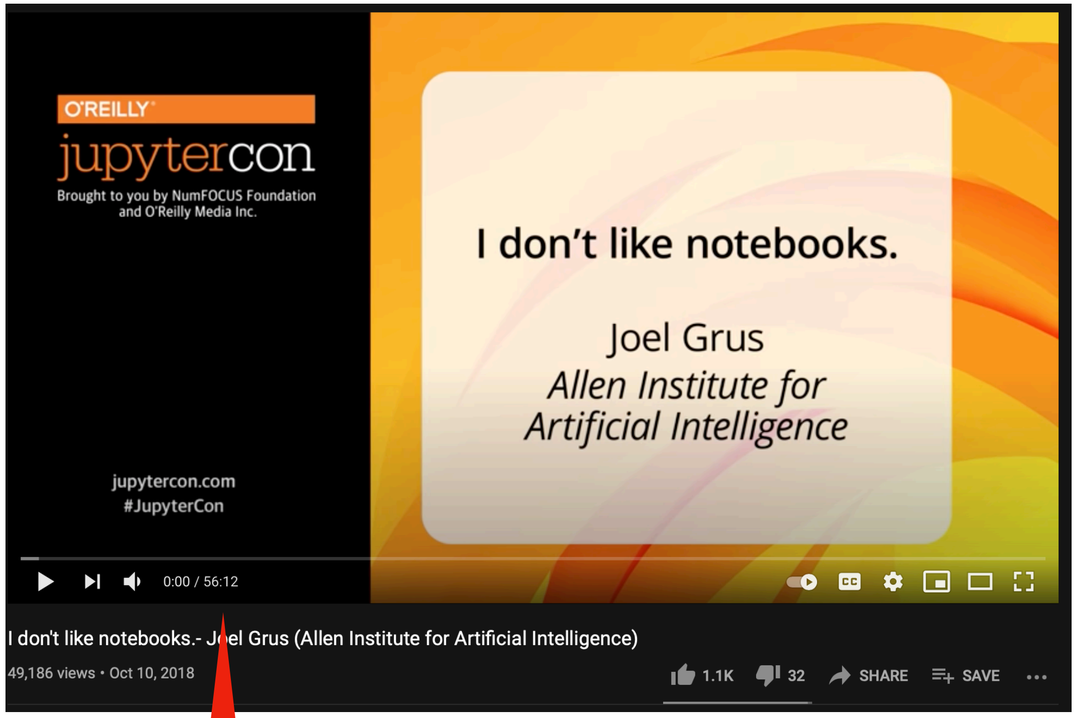
What's the First Thing You Do?


```
print("Hola World!!")
```

```
Hola World!!
```

Why Should You Avoid Notebooks

- Running code out of order seems like a good feature, but it’s not.
- Don’t lose the features what a great IDE (such as VSCode, PyCharm or Sublime) can offer you.



56 minutes of uninterrupted rant against Notebooks

https://www.youtube.com/watch?v=7jiPeIFXb6U

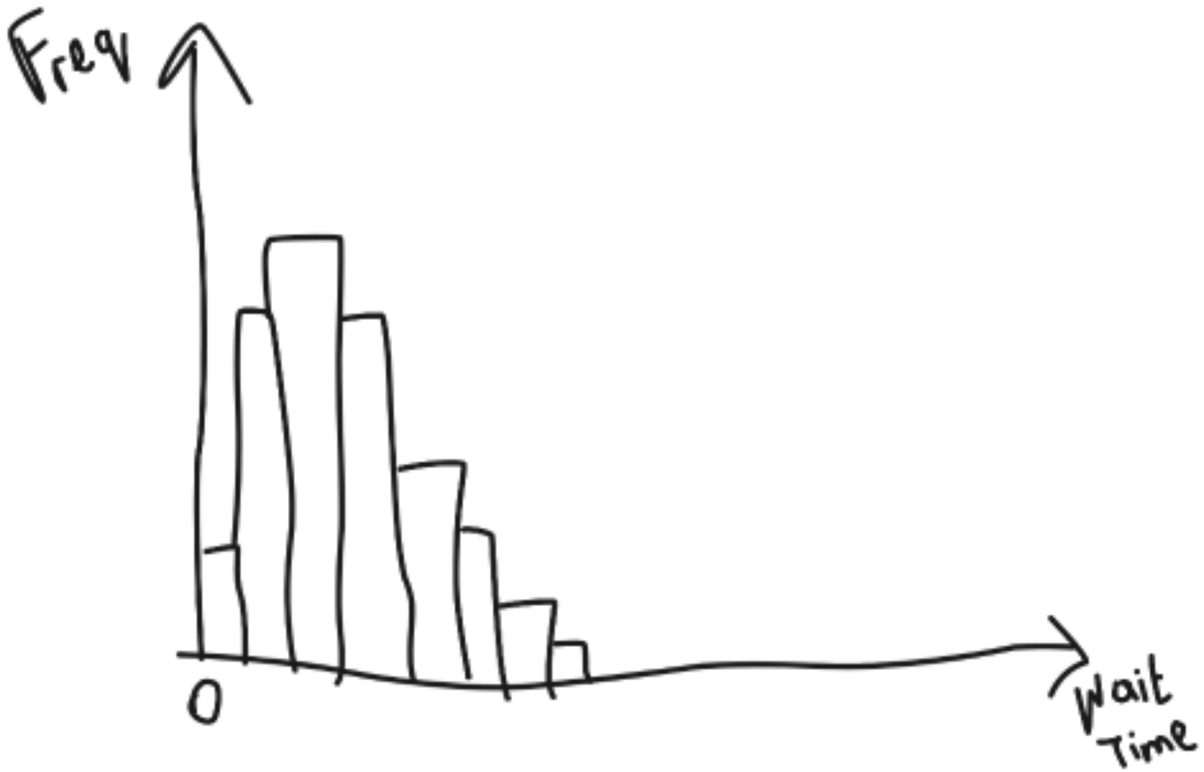
Lessons So Far

Avoid Jupyter Notebooks
until you cannot.

Where to Start Coding First From?

The YAGNI Principle

How Will the End Product Look Like?



End to Start

Task: Plot the histogram of wait times (for a given lambda, mu and a simulation run-length)

*Here be Dragons

→

Plot Histogram Of Wait Time Data

How I Would Start

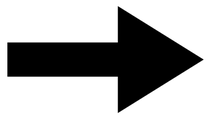
mm1queue.py > ...

```python
import seaborn as sns
import matplotlib.pyplot as plt

def plot_wait_time_histogram(wt_data):
    return sns.histplot(wt_data, x="WaitTime")

if __name__ == "__main__":
    wt_data = ... # Here be Dragons
    ax = plot_wait_time_histogram(wt_data)
    plt.show()
```

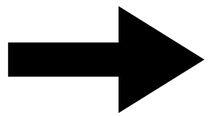
End to Start

Task: Plot the histogram of wait times (for a given lambda, mu and a simulation run-length)

*Here be Dragons



Obtain Data to Plot



Plot Histogram Of Wait Time Data

Moving On

mm1queue.py > ...

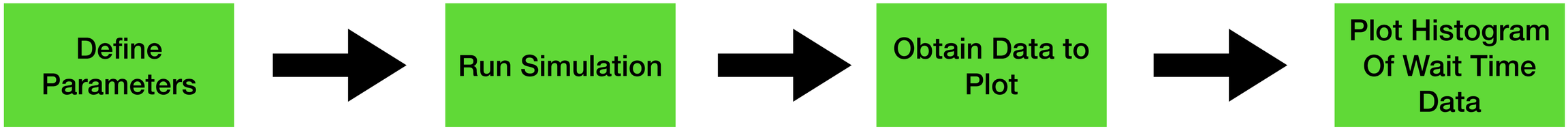
```python
import seaborn as sns
import matplotlib.pyplot as plt

def plot_wait_time_histogram(wt_data):
    return sns.histplot(wt_data, x="WaitTime")

if __name__ == "__main__":
    simulation = ... # Now we pushed the dragons here
    wt_data = simulation.get_wait_time_data()
    ax = plot_wait_time_histogram(wt_data)
    plt.show()
```

End to Start

Task: Plot the histogram of wait times (for a given lambda, mu and a simulation run-length)

End to Start

Task: Plot the histogram of wait times (for a given lambda, mu and a simulation run-length)



Done with the Skeleton

mm1queue.py > ...

```python
import seaborn as sns
import matplotlib.pyplot as plt

def plot_wait_time_histogram(wt_data):
    return sns.histplot(wt_data, x="WaitTime")

if __name__ == "__main__":
    lmb = 3
    mu = 5
    run_length = 1000
    simulation = MM1QueueSimulation(lmb=lmb, mu=mu)
    simulation.run(run_length=run_length)
    wt_data = simulation.get_wait_time_data()
    ax = plot_wait_time_histogram(wt_data)
    plt.show()
```

Implementing the M/M/1 Queue

$a_i \sim Exp(\lambda)$ $\quad a_1 \quad a_2 \quad a_3 \quad \dots$

Generate inter-arrival times

$a_1 \quad a_1 + a_2 \quad a_1 + a_2 + a_3 \quad \cdots \quad a_1 + a_2 + a_3 + \ldots > T_{max}$

1st Arrival

2nd Arrival

3rd Arrival

Stop when later than run length

Implementing the M/M/1 Queue

$a_i \sim Exp(\lambda)$

$s_i \sim Exp(\mu)$

Generate Service Times, the same way you did for arrival times

a_1	$a_1 + a_2$	$a_1 + a_2 + a_3$	$\cdots$	$a_1 + a_2 + a_3 + \ldots > T_{max}$

s_1	$s_1 + s_2$	$s_1 + s_2 + s_3$	$\cdots$	$s_1 + s_2 + s_3 + \ldots > T_{max}$

Implementing the M/M/1 Queue

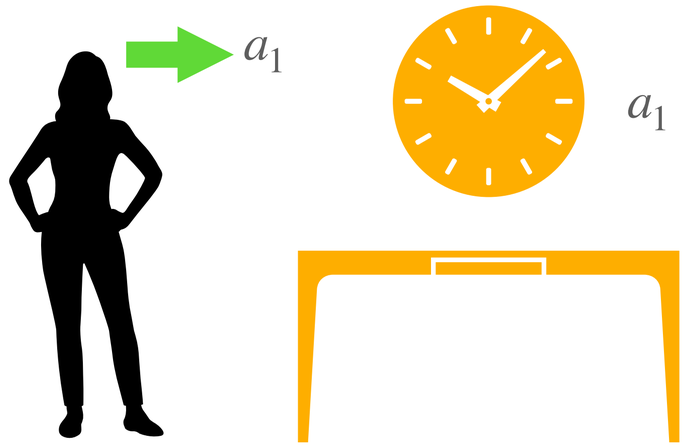
$a_1 < s_1$

Next event is an arrival.

a_1	$a_1 + a_2$	$a_1 + a_2 + a_3$	$\cdots$	$a_1 + a_2 + a_3 + \ldots > T_{max}$
s_1	$s_1 + s_2$	$s_1 + s_2 + s_3$	$\cdots$	$s_1 + s_2 + s_3 + \ldots > T_{max}$

Implementing the M/M/1 Queue

$a_1 < s_1$

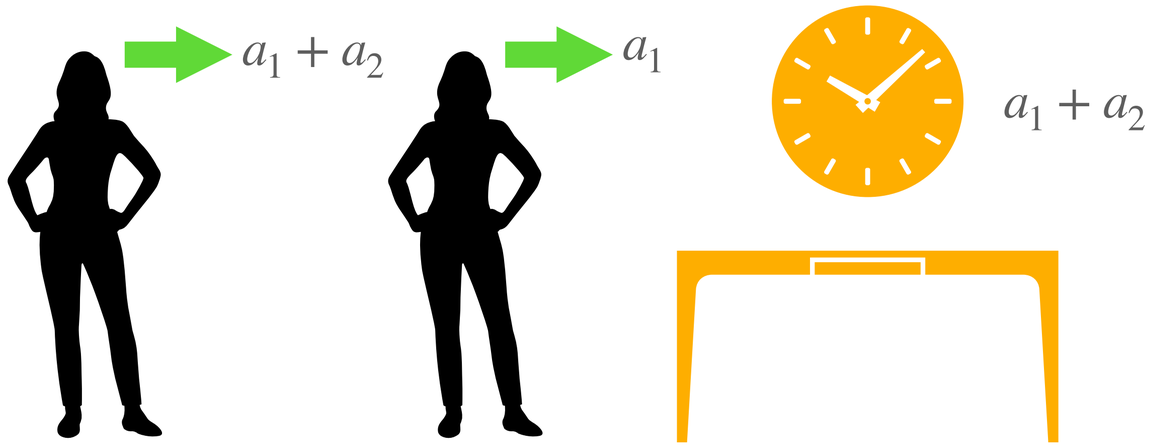


a_1	$a_1 + a_2$	$a_1 + a_2 + a_3$	$\cdots$	$a_1 + a_2 + a_3 + \ldots > T_{max}$

s_1	$s_1 + s_2$	$s_1 + s_2 + s_3$	$\cdots$	$s_1 + s_2 + s_3 + \ldots > T_{max}$

Implementing the M/M/1 Queue

$a_1 + a_2 < s_1$

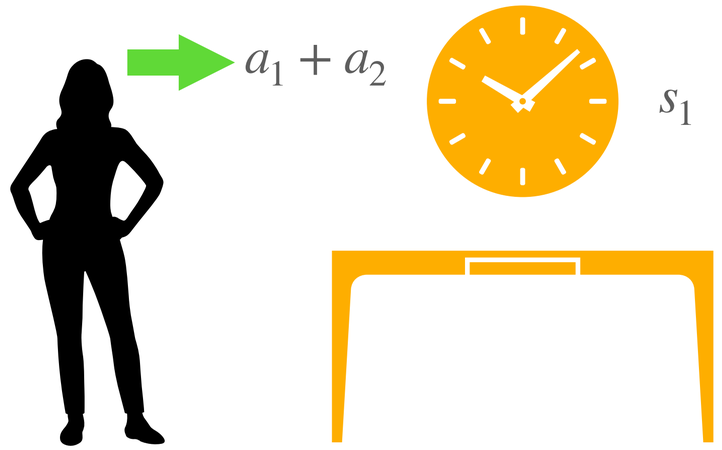


	$a_1 + a_2$	$a_1 + a_2 + a_3$	$\cdots$	$a_1 + a_2 + a_3 + \ldots > T_{max}$

s_1	$s_1 + s_2$	$s_1 + s_2 + s_3$	$\cdots$	$s_1 + s_2 + s_3 + \ldots > T_{max}$

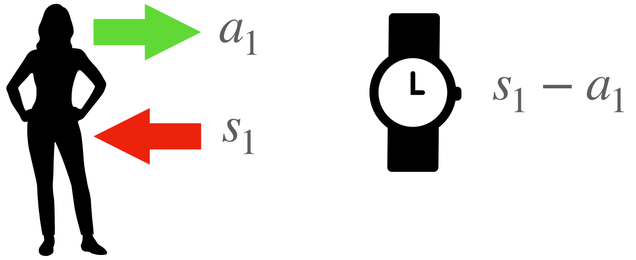
Implementing the M/M/1 Queue

$$a_1 + a_2 + a_3 > s_1$$

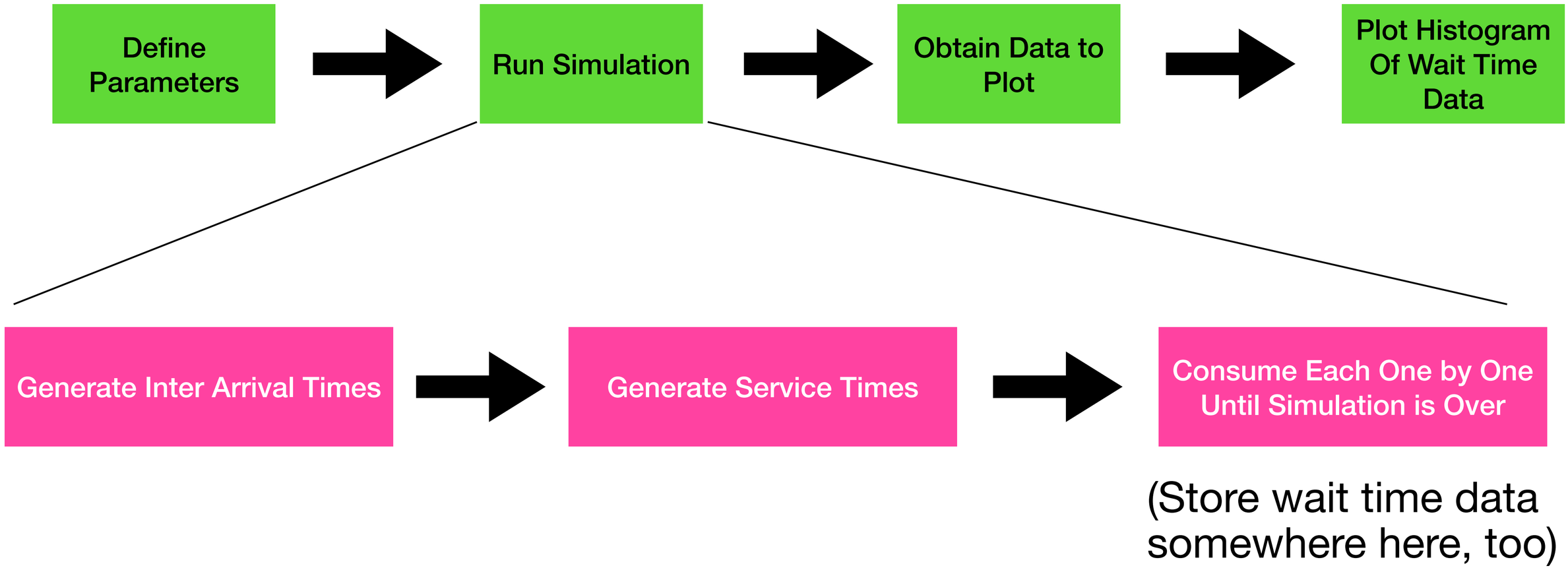


	$a_1 + a_2 + a_3$	$\cdots$	$a_1 + a_2 + a_3 + \ldots > T_{max}$

s_1	$s_1 + s_2$	$s_1 + s_2 + s_3$	$\cdots$	$s_1 + s_2 + s_3 + \ldots > T_{max}$

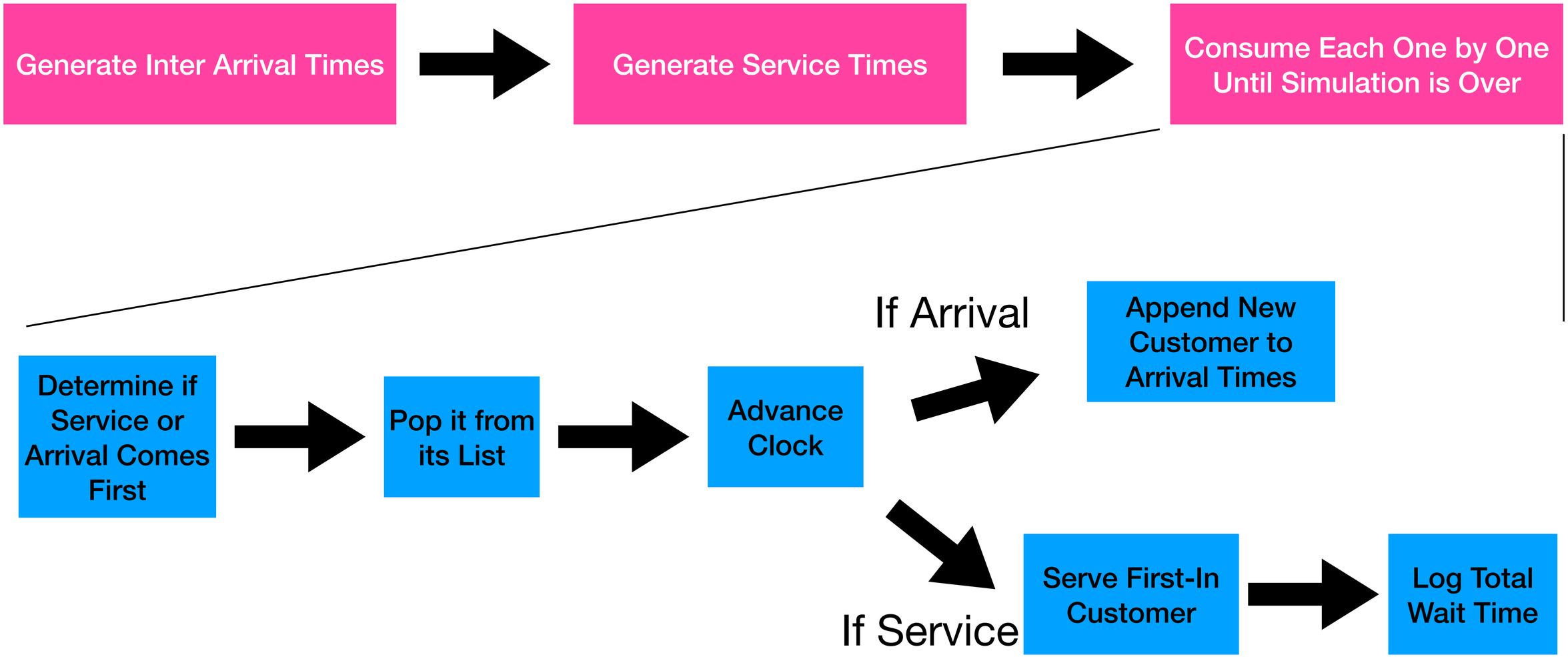


Now, Top to Bottom



(Store wait time data somewhere here, too)

Getting Deeper…



Check the code…

Why End to Start, Top to Bottom?

- **End to Start:**
  - Don’t implement anything you won’t need.
- **Top to Bottom:**
  - Divide and conquer.
  - Small steps are easier to implement, read and test.

Lessons So Far

Avoid Jupyter Notebooks
until you cannot.

End to Start, Top to Bottom

Document early, often

```
def consume_next_event(self):
    """Issue arrival or service, whichever comes earlier
    """
    next_event, next_event_time = self.pick_next_event()
    # Advance Clock
    self.clock = next_event_time
    if next_event == "Arrival":
        self.append_new_customer_to_wait_times(arrival_time=next_event_time)
    else:
        first_in_customer_arrival_time = self.serve_first_in_customer()
        first_in_customer_wait_time = self.clock - first_in_customer_arrival_time
        self.wait_times.append(first_in_customer_wait_time)
```

This is a Python Docstring

This is a Python Comment

Check the code…

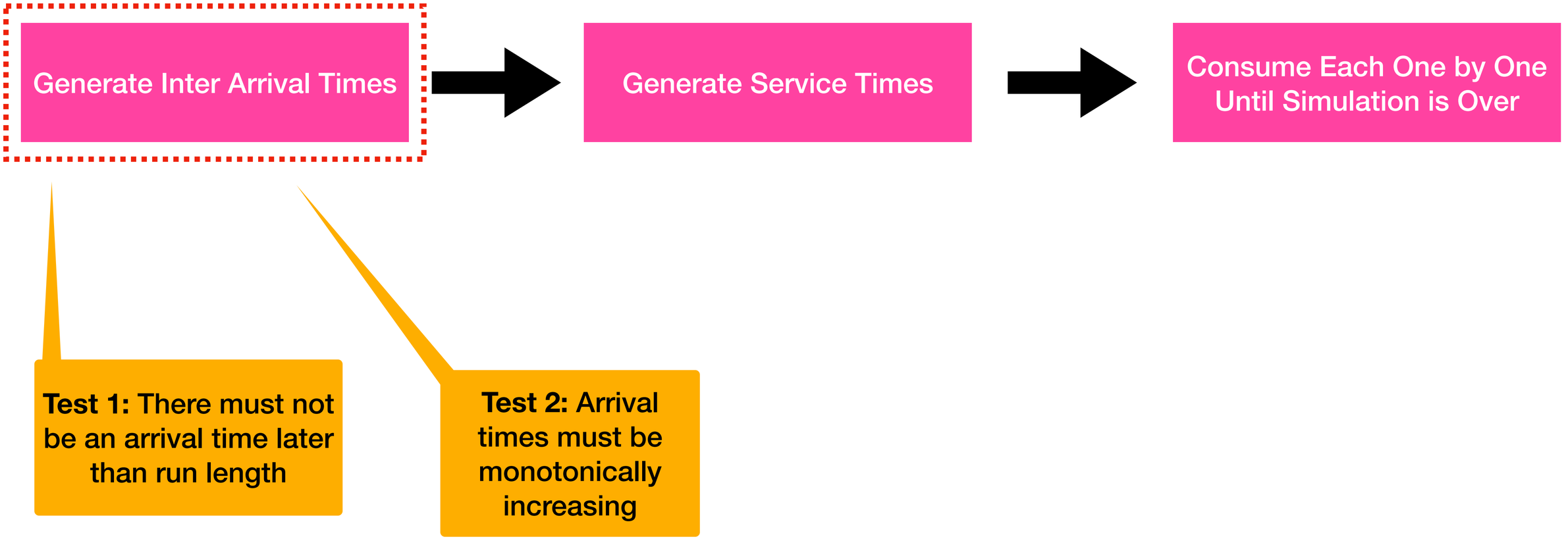
Lessons So Far

Avoid Jupyter Notebooks until you cannot.

Document early, often

End to Start, Top to Bottom

Test First, Test Everything



How Testing Helps

- Forces you to think about, and cover for corner cases.
- Catch errors early, not after your ran the code and waited for an hour.
  - Saves headaches.
- You'll get faster at writing code (quality > quantity).
- BONUS: builds fundamental skills for coding challenges.

Let’s go through the updated tests…

Lessons So Far

Write tests before implementations

Avoid Jupyter Notebooks until you cannot.

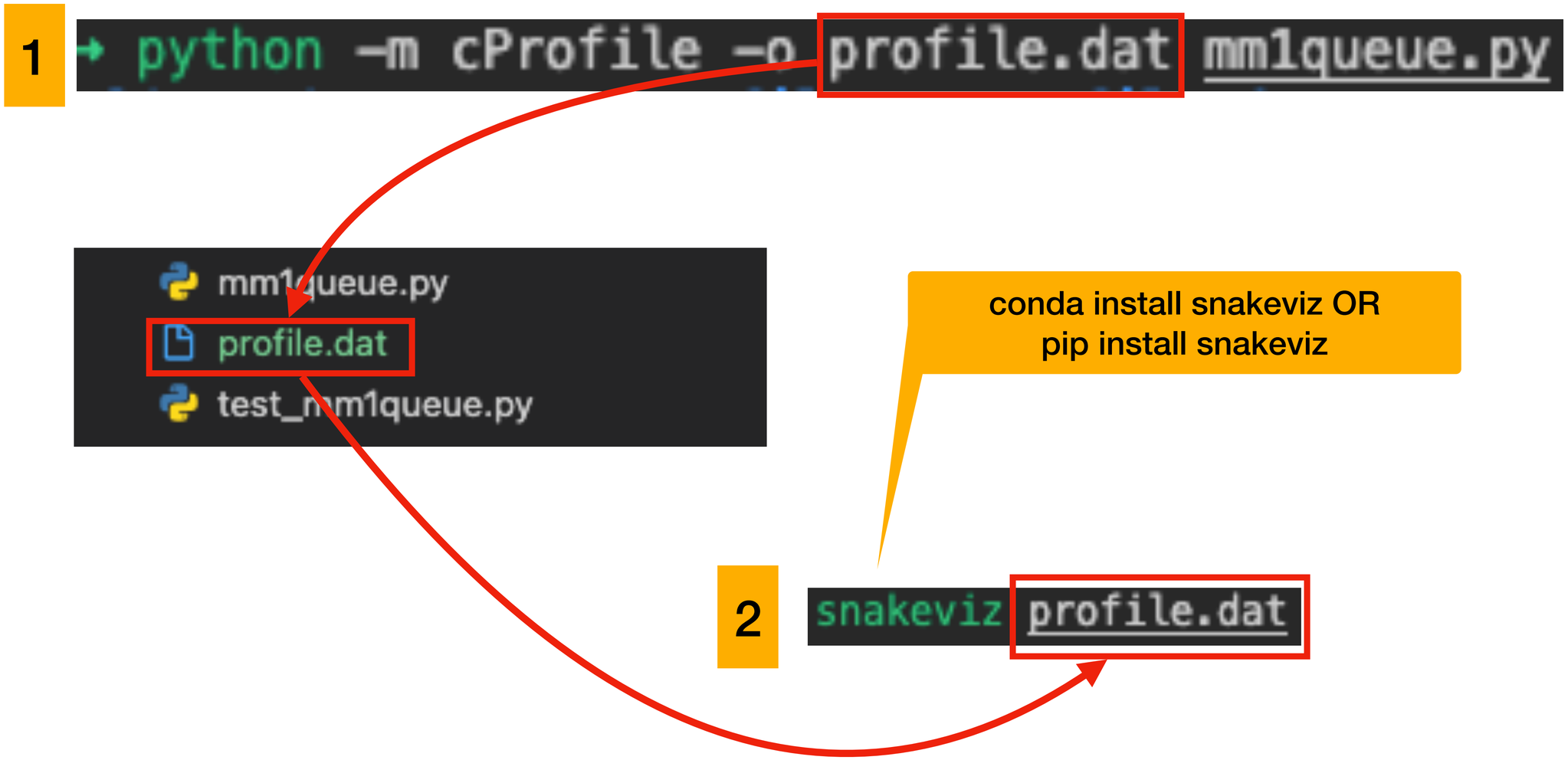
Document early, often

End to Start, Top to Bottom

How to Scale?

- When you need to scale up your experiments:
  - Find the bottlenecks in your code.
  - Speed them up.

Basic profiling, in 2 steps

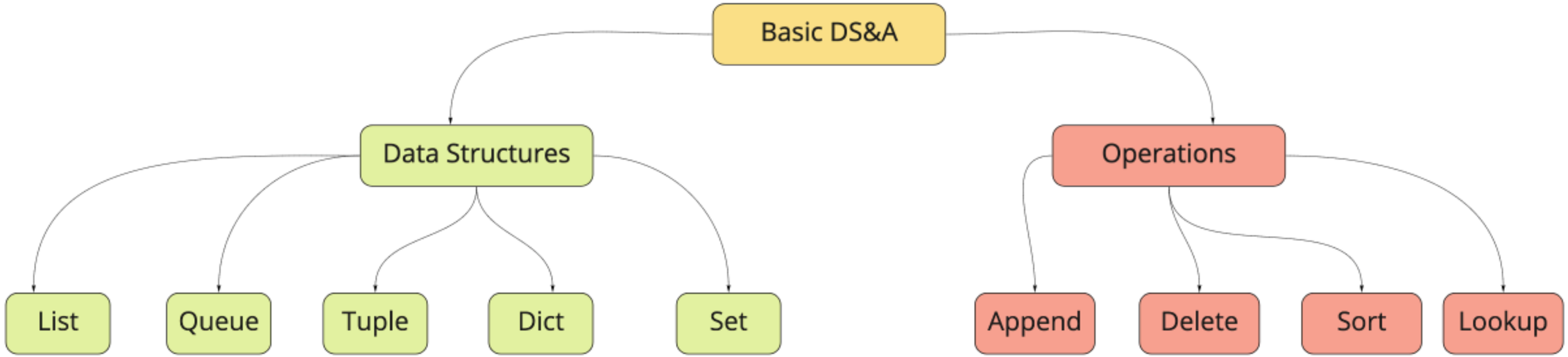


Profiling in Action…

Learn Data Structures and Algorithms

- You don’t have to be as good as a software engineer.
- Pros:
  - You’ll organize information more effectively.
  - You’ll know better how to speed up your code when you have to.

What to Know


Basic DS&A
Data Structures
Operations
List
Queue
Tuple
Dict
Set
Append
Delete
Sort
Lookup

Lessons So Far

Learn Basic Data Structures and Algorithms

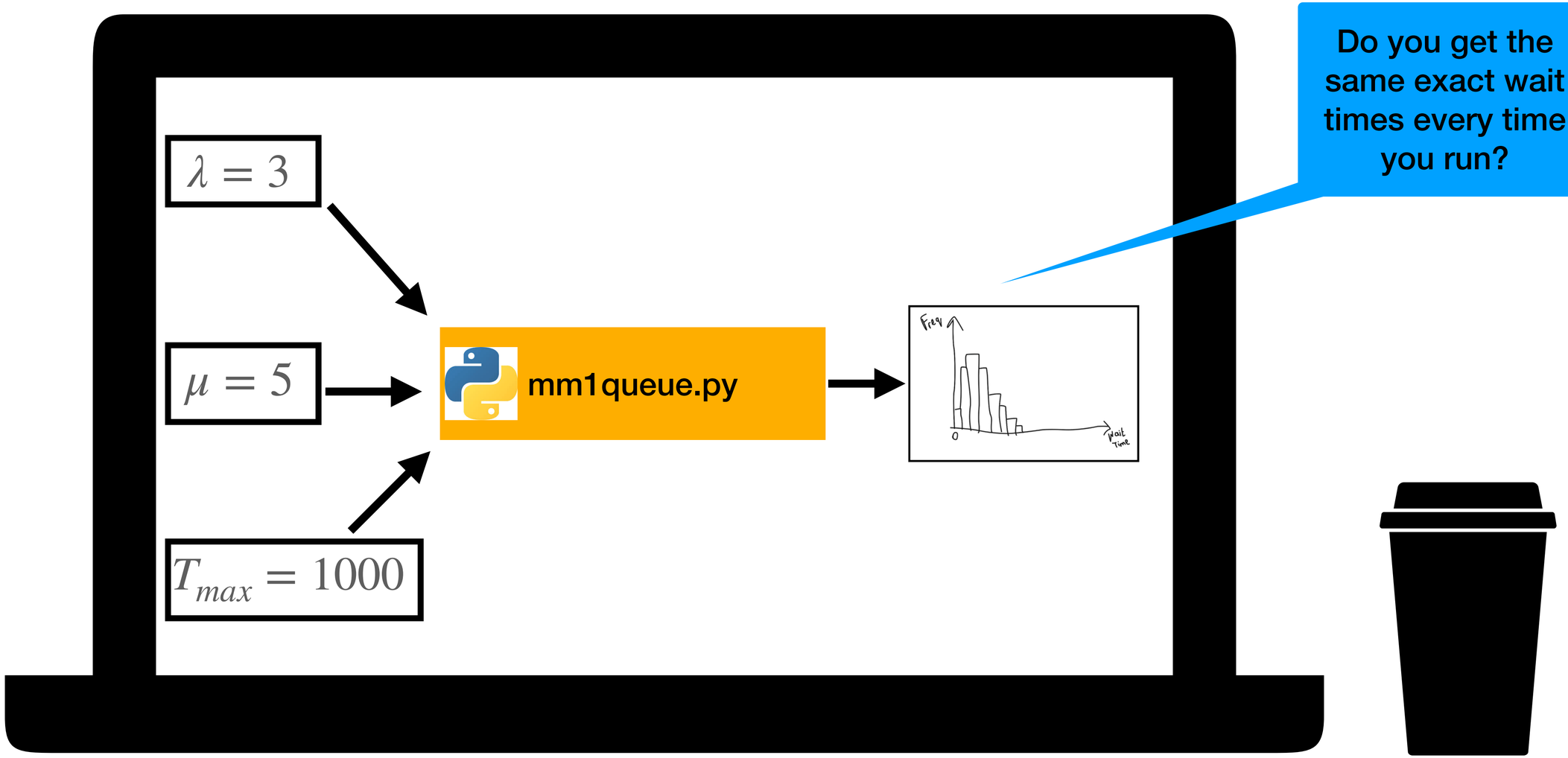
Write tests before implementations

Avoid Jupyter Notebooks until you cannot.

Document early, often

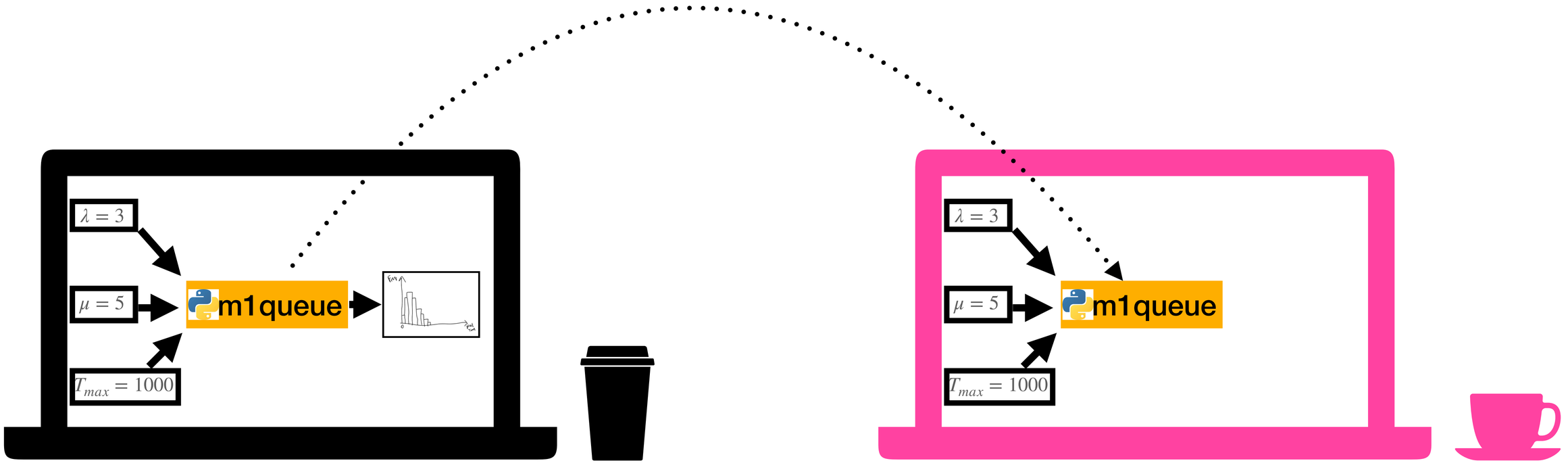
End to Start, Top to Bottom

Write Reproducible Code

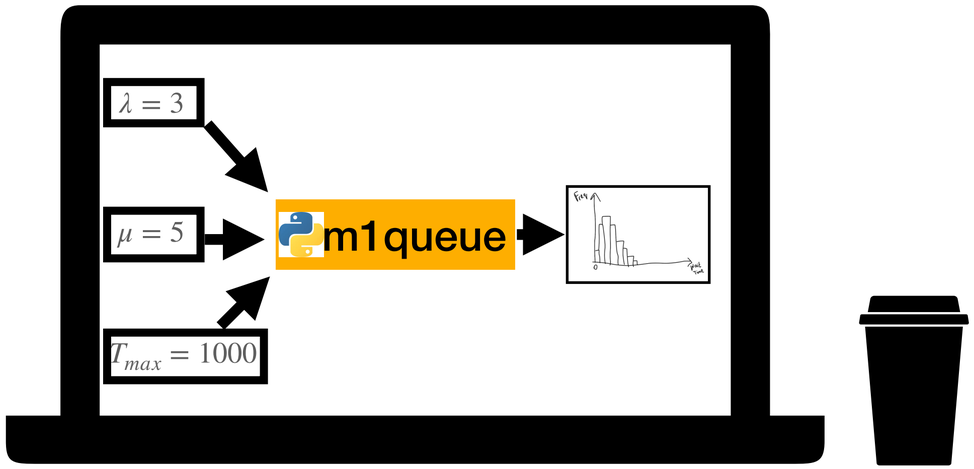


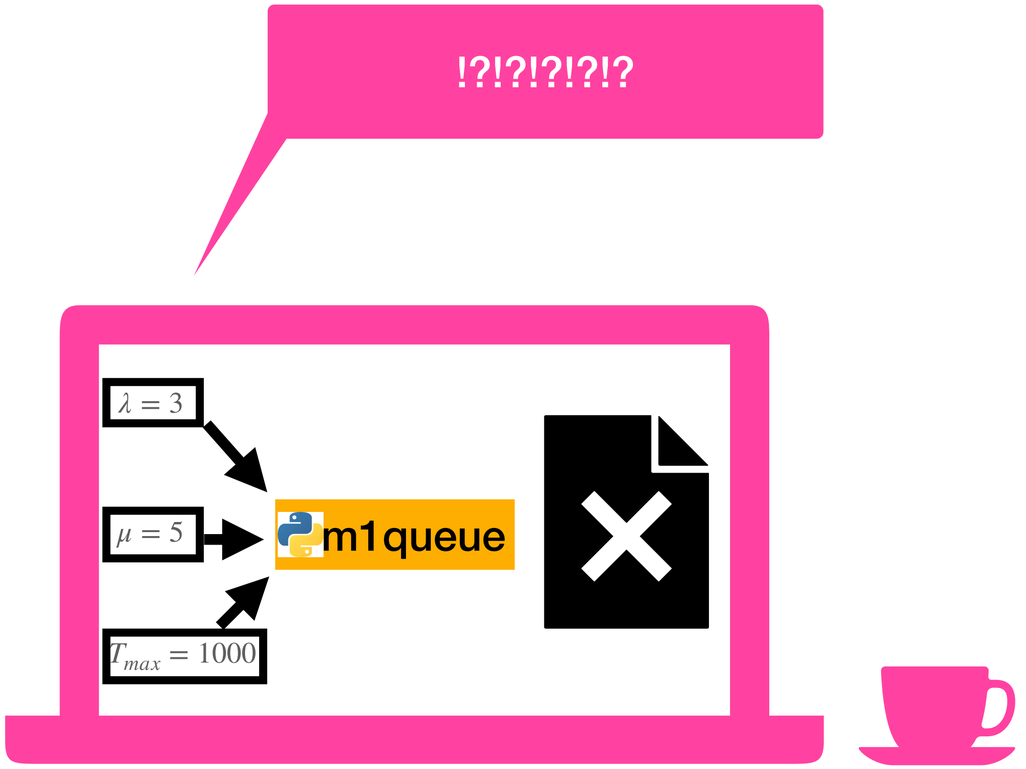
Write Reproducible Code

You email the code to your colleague.

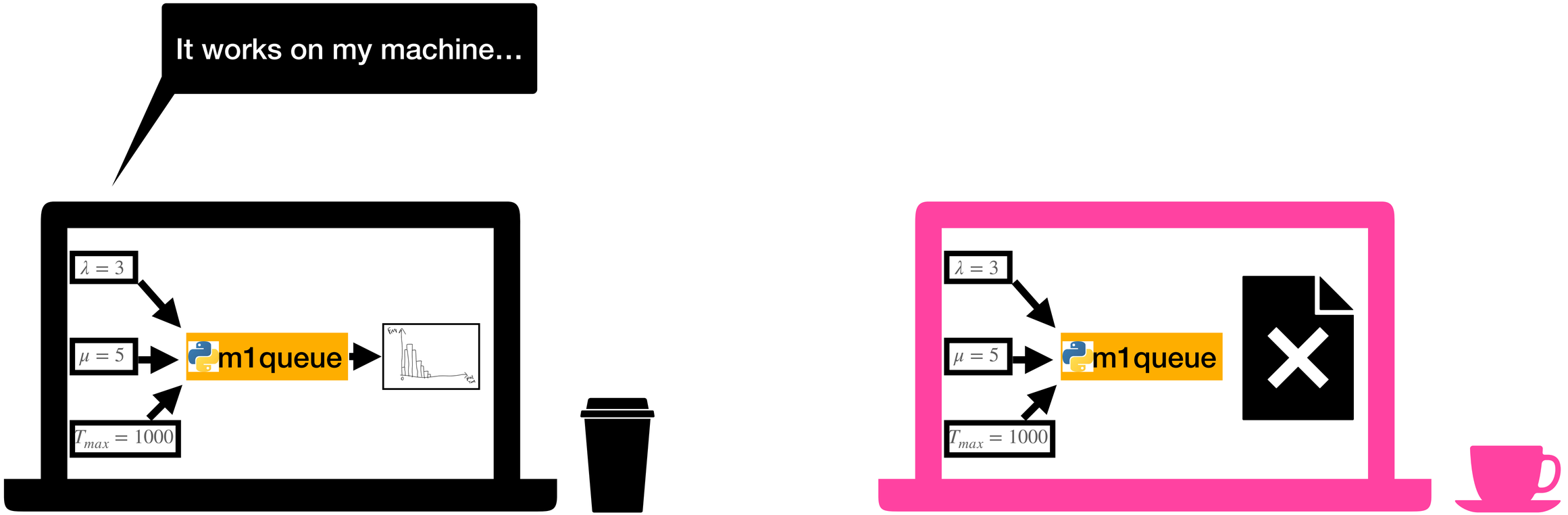


Write Reproducible Code

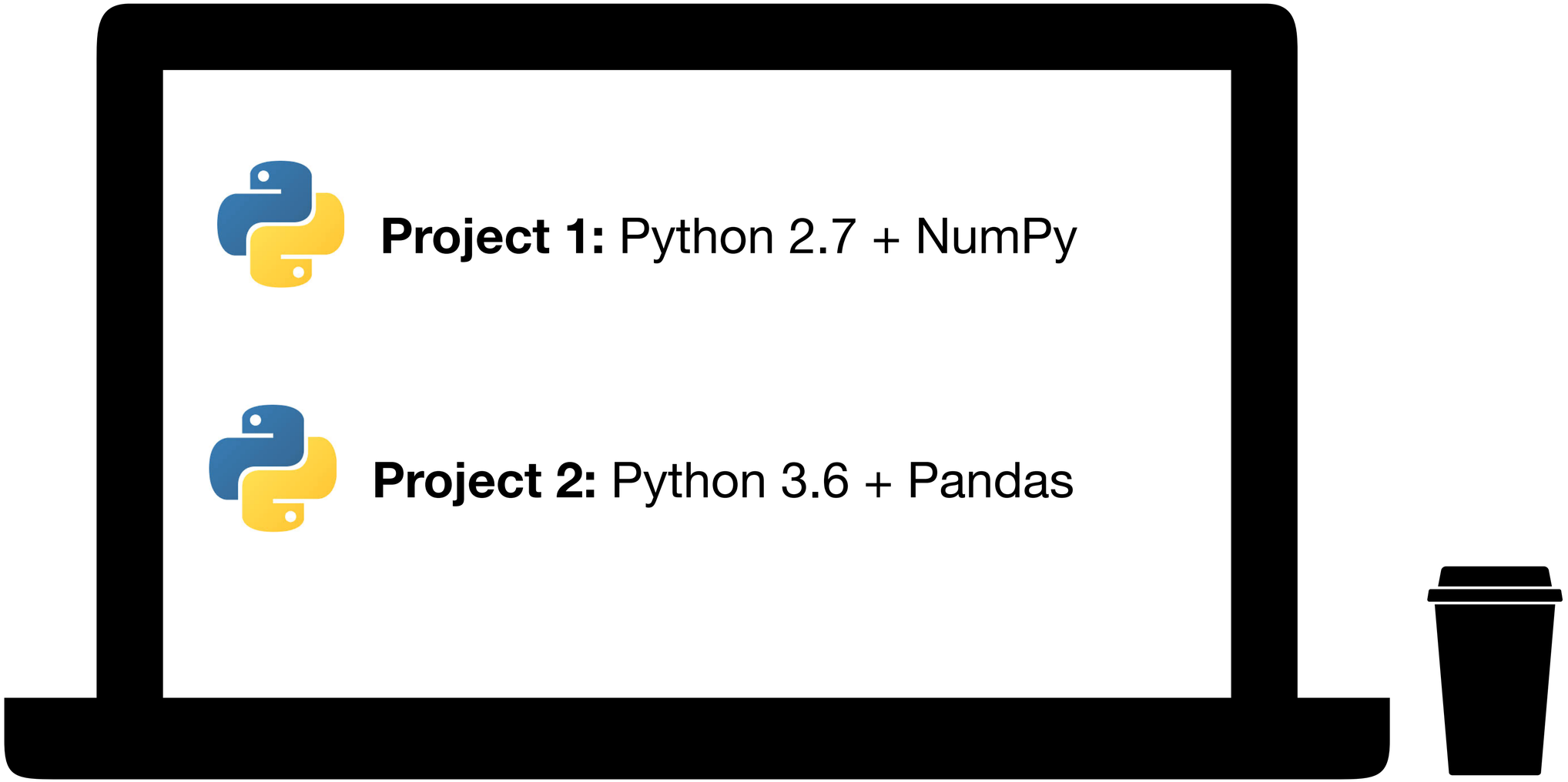




Write Reproducible Code



Virtual Environments



Virtual Environments

Python has many tools to handle virtual environments.

I recommend *venv*.

Conda Has Environments.

I suggests going through the documentation.

Lessons So Far

Learn Basic Data Structures and Algorithms

Write tests before implementations

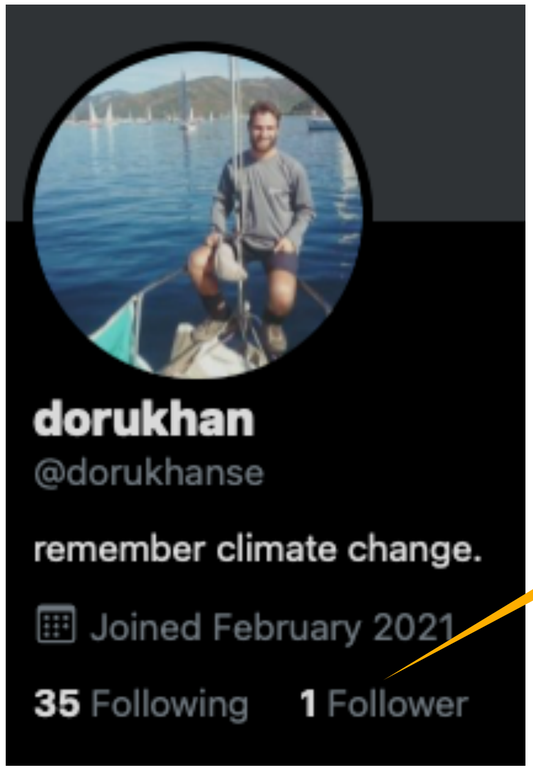
Avoid Jupyter Notebooks until you cannot.

Document early, often

End to Start, Top to Bottom

Write Reproducible Code

Thank You!



Quick! Be my second follower!

https://twitter.com/dorukhanse