Python Prework

IMC

August 2023

CONTENTS

CHAPTER

ONE

INTRODUCTION

I hope you are as excited as we are about you joining IMC, and we're looking forward to having you in our Software Engineering Global Traineeship (colloquially known as Dev School).

Before we start the traineeship together, we want to ensure that you have a solid understanding of certain widely used open-source technologies. We have a limited amount of time available for the traineeship, so we want to be able to use that time for working directly with instructors to learn advanced topics and IMC-specific tooling. Therefore, it is essential that you spend time familiarizing yourself with these technologies prior to the start of the program. The attached documents describe the contents that you should focus on.

While we will provide you with recommended learning resources as a starting point, please note that independent research and exploration are expected from each of you. Self-directed learning is a crucial skill for success at IMC, and this is an opportunity to practice that!

For each technology, we have compiled a set of recommended learning resources that will serve as a great starting point for your studies. We have found these to be good introductory resources for covering fundamental concepts and providing practical examples to help grasp the essentials. If they do not fit your learning style, you are encouraged to find your own resources on Google, YouTube, or elsewhere. If you are already familiar with any material, feel free to skip the specific content.

Additionally, we are providing you with a checklist of concepts for each technology. This checklist will serve as a valuable tool to ensure that you cover the most important topics within each technology. It is highly recommended that you use this checklist as a guideline to gauge your progress and track your learning.

It is very important that you are completely comfortable with these basics, as they are mostly taken for granted in the training.

If you have any questions or need further clarification, please don't hesitate to reach out to us. We are more than happy to assist you.

CHAPTER

TWO

GIT

Git is the dominant VCS in the tech community, and is used by all development teams at IMC.

Additionally, most teams use the Gerrit code review tool, which heavily relies on advanced Git features around rewriting history.

2.1 Recommended Reading Material

2.1.1 Pro Git

The following chapters contain the most useful material about using Git.

- 1. Getting Started
- 2. Git Basics
- 3. Git Branching
- 7. Git Tools
  - 7.1 Revision Selection
  - 7.2 Interactive Staging
  - 7.3 Stashing and Cleaning
  - 7.5 Searching
  - 7.6 Rewriting history
  - 7.7 Reset Demystified
  - 7.10 Debugging with Git

Chapters 4-6 cover Git Servers and Workflows that are applicable to working in open source and on Github, but not immediately relevant to work at IMC.

The other sections of Chapter 7 cover more advanced Git tools that are useful, but will not be used in Dev School.

2.2 Recommended Exercises

Learn Git Branching is a good practical exercise for learning about the most useful Git commands.

2.3 Checklist

2.3.1 Essential Concepts

You should have a solid understanding of the following concepts.

- Git Fundamental Elements
  - Commits
  - Branches
  - Tags
  - References
  - Remotes
  - Repositories
- Commit Workflow
  - Working Directory
  - Stage/Index
  - Stash
- Checking out history
- Rewriting history
- `git reset`

2.3.2 Practical Skills

You must be able to do the following:

- Git commit workflow
  - Stage change and create commits
  - See your uncommitted or staged changes
  - Throw away your un-committed changes
  - Stash and unstash changes, stash multiple changes at a time
- Exploring Git History
  - See the history of commits in a repository
  - See the commit message for any given commit?
  - See what changes went into any given commit, or what are the differences between two arbitrary commits
  - Look at the code from a previous state in the repository history
  - Figure out what commit a bug was introduced in

- Branching and tagging
  - Create, switch, merge and delete branches
  - Copy commits from one branch to another
  - Create and delete tags
- Git Remotes
  - Clone from, pull from, push to a remote
- Undoing
  - Undo a `git commit` (without losing your code).
  - Revert a commit so that the code is in the same state as if the commit wasn't done, but the commit is left in history.
  - Delete a commit so that it is gone from git history entirely.
- Rewriting history
  - Modify the commit message of the most recent commit.
  - Modify the commit message of a previous commit
  - Combine two commits into one commit
  - Split one commit into two commits
  - Reorder two commits
  - Delete a commit
  - Change a branch to point to a completely different commit
- Fixing mistakes
  - Undo a `reset`
  - Recover a "lost" commit

2.3.3 Advanced Git Concepts and Tools

You are **not** required to know anything about these.

- Submodules / Subtrees
- Sparse Checkouts
- Hooks
- Git internals
  - Objects: Blobs, Trees, etc.
  - Packfiles
- `git rerere`

2.4 Additional Material

- An Introduction To Git and Github - Introduction to Git, but in video format.
- Think Like (a) Git

2.5 Exam

The *Git Exam* can be used to evaluate your understanding of git.

This exam is not graded - you should it to identify weaknesses in your understanding.

2.5.1 Git Exam

Theory

1. What does it mean when Git commits are described as immutable? How is that reconciled with the ability to amend or delete commits?
2. What does it mean to merge branches A into branch B?
3. What does it mean to rebase branch A on top of branch B?
4. What is a "fast-forward merge"?
5. How does a tag differ from a branch?
6. What are the most common use cases for tags?
7. What is a "commit-ish"?
8. What is the difference between `HEAD` and a "head"?
9. What is the `origin`?
10. What is the different between `git fetch` and `git pull`?
11. What is a merge conflict? How do you resolve a merge conflict?
12. How would you find the commit where a bug was introduced?
13. How do you check out the code for a previous commit so that you can debug it?
14. What is "Detached HEAD mode"? How do you get into detached HEAD mode? How do you get out of detached HEAD mode?
15. What does rewriting history mean? When might you do it? When should you not rewrite history? What can go wrong when you modify history?
16. When should you delete a commit vs revert it?
17. What is the difference between `git reset 29f5e54` and `git checkout 31f5e54`?
18. What is the difference between a soft, hard, and mixed reset?
19. When would you use `git reset`?
20. Is it possible to lose a commit? Or to lose your changes? Under what circumstances?
21. What is the `reflog`? When would you use it?

Practical

Please clone https://github.com/imc-trading/devschool-git-exam.

1. Check out the branch `resume`. Change the commit message from "Add Wok Experience" to "Add Work Experience". What commands did you run?
2. Add this "Work Experience" item in "Resume.md":

```
### The Normal Brand

- Web Developer -- 2019-2021
  - Implemented OAuth login process to support Social Login.
  - Added web analytics
```

Commit it on the branch `resume` with the message "Add Normal Brand item". What command(s) did you run?

3. The commit on the branch `resume` with the message "fill edu" has some issues. Fix it like this: 1. Change the message to "Add content for Education section". 2. The commit has introduced some whitespaces at the end of two lines. Remove those. What command(s) did you run? Explain your actions.
4. The commit with the message "Add empty lines" is not useful. Remove it from the history. What command(s) did you run?
5. There is a branch called `skills`. How can you show the content of the `Resume.md` file on that branch without switching to the branch?
6. On the branch `skills` there is a commit called "Add skills". Add that commit on top of the branch `resume` without switching branches away from `resume`. That should create a conflict. Fix it by incorporating both the previous and the new changes. What commands and actions did you do?
7. In `Resume.md` there is a line `### Illinois State University`. How can you find out what is the last commit that changed/introduced that line? What is the message of that commit? What commands did you run?
8. Create a branch called `letter_of_intent` forking off from the branch `job_applications`. Create a file there called `letter.txt` with the content `I need this job, please!`. Commit this file on the `letter_of_intent` branch with the commit message `Add letter of intent`. Add another commit that adds this line to `letter.txt`: `Sincerely yours, Norman`. Commit it with the message `Sign letter`. What commands did you run?
9. Rebase the `letter_of_intent` onto the `resume` branch without including the commits on the `job_applications` branch. Basically add the commits `Add letter of intent` and `Sign letter` on top of `resume` without changing `resume` and point `letter_of_intent` to the last commit. Do not use `cherry-pick`. What command did you run?
10. Merge the branch `letter_of_intent` into the branch `resume`. What commands did you run? Explain the merge strategy taken.
11. Merge the `job_applications` branch into the branch `resume`. What commands did you run? What merge strategy was taken? What happened to the git history of the `resume` branch?

CHAPTER

THREE

UNIX BASICS

All of our deployment servers are UNIX environments and you will need to be able to interact with them using the command line in order to effectively deploy and manage your apps at IMC.

3.1 Checklist

3.1.1 Essential skills

You must be able to do the following:

- Be familiar with the UNIX command line interface and its basic functionalities.
  - Be able to navigate through a file system, create and remove files and directories. Use the `cd`, `ls`, `mkdir`, `cp`, `mv` and `rm` commands.
- View files using `cat` and `less`
- Edit files using a CLI text editor.
  - We recommend `vim` but `emacs` and `nano` are also available.
    - `vimtutor` is a useful tool for learning.
  - Be able to open, edit and write files from the CLI.

3.1.2 Useful skills

The following skills are extremely useful, but not critical for Dev School.

- Search through files and directories for text patterns with `find` and `grep`
- Redirect output to a new command with the `|` operator.
  - Combine this with `grep`!
- Redirect output to a file with > and >>
- Understand environment variables, `env`, `export`, `echo`
- Monitor a system with `ps`, `top` and `htop`
- Manage processes (`kill`)

3.1.3 Bonus

- Use a CLI editor to create a simple bash script file which prints "Hello World" (`echo`).
- Make it runnable (`chmod +x`) and run it directly from the command line.
- If your script file is called `hello.sh` what is the difference between running `bash hello.sh` and `./hello.sh`? What is a shebang?

CHAPTER

FOUR

DOCKER

Containers streamlines the process of deploying production services by enabling developers to package an application with all its dependencies into a standardized unit for software development. This approach, known as containerization, ensures that the application will run uniformly, regardless of any customized settings that machines might have that could differ from the machine used for writing and testing the code.

Docker is IMC's containerization tool of choice. Docker's lightweight, scalable, and consistent environment is particularly beneficial for creating, deploying, and running applications in a variety of environments, from local development machines to production servers, thereby increasing productivity and reducing overhead.

4.1 Recommended Tutorial

- Docker - Getting Started
  - Parts 1-9

4.2 Checklist

4.2.1 Essential Concepts

You should have a solid understanding of the following concepts. We've included some follow-up questions and related concepts under each point.

- Containers
  - What is a container?
  - Difference between virtual machines and containers
  - Advantages of containerization
  - What is Docker?
- Docker Images
  - What is a Docker image?
  - Dockerfile structure and commands
    - FROM, RUN, CMD, ENTRYPOINT, ENV, COPY, ADD, etc
  - Docker image layers
  - Image tags

- Docker Containers

4.2.2 Useful Concepts

You should have a general familiarity with the following concepts, but deep understanding is not necessary.

- Docker Volumes and Bind Mounts
  - Persistent data and Docker storage options
  - Docker volumes
  - Bind mounts
- Docker Networking
  - Host networking
  - Port binding
- Docker Best Practices
  - Dockerfile best practices
  - Docker development best practices

4.2.3 Practical Skills

You must be able to do the following:

- Build images using a Dockerfile
- Manage and delete images
- Create, start, stop, and remove containers
- Interact with running containers
- Read container logs
- Execute commands in a running container
- Inspect and monitor Docker containers
- Manage and delete containers
- Mount a local directory into a docker container
- Bind ports

4.3 Exam

The *Docker Exam* can be used to evaluate your understanding of git.

This exam is not graded - you should it to identify weaknesses in your understanding.

4.3.1 Docker Exam

1. What command(s) do you use to run MariaDB in Docker on your machine and map its port to the port `12345` on your machine? It should contain a database called `prework`.
2. What command(s) do you use on your local machine to open a prompt into the MariaDB server from step 1? You can use any Mysql client.
3. Clone `https://github.com/imc-trading/devschool-docker-exam`
4. How do you run step 1 but use `init.sql` to initialize the database?
5. What command do you use to open bash console in your running MariaDB container?
6. What commands do you use to list containers, list images, stop a container, start a container, remove a container?
7. In `server.py` you have a small Python server implementation. Please create a custom Docker image around it and a Docker compose that starts up this server and the MariaDB container from the steps above. The server should connect to the MariaDB container and it should expose its HTTP port to localhost's port 8082. How do you check that the server works? Note that the server has requirements that must be installed with `pip install -r requirements.txt`

CHAPTER

FIVE

KUBERNETES AND HELM

Kubernetes is a powerful tool for orchestrating containerized applications, offering automated management, scaling, and updating of applications across multiple hosts.

Helm is a package manager for Kubernetes that simplifies deployment of applications. It uses a packaging format called Charts, which are collections of files that describe a related set of Kubernetes resources. Helm provides a way to manage and maintain complex applications through a simple, declarative approach, making deployments repeatable and more manageable.

The next generation of IMC's deployment tools are built on top of Kubernetes and Helm, so it is valuable to have a basic understanding of how these tools work and how to work with them.

Note: Kubernetes is a complex topic with a lot of depth, most of which you don't need. Just make sure you know the K8s and Helm concepts listed in the *checklist below*. The recommended readings and tutorials are good places to start.

5.1 Resources

5.1.1 Recommended Reading

- Introduction to Kubernetes
- Config Maps

5.1.2 Recommended Tutorials

- Hello Minikube
- Helm Chart Tutorial

5.1.3 Alternatives

- Youtube - Kubernetes Crash Course

5.1.4 Reference documentation

- Kubernetes Documentation
- Helm Documentation

5.2 Checklist

5.2.1 Essential Concepts

You should have a solid understanding of the following concepts. We've included some follow-up questions and related concepts under each point.

- Kubernetes Basics
  - What is Kubernetes?
  - What are the benefits to using Kubernetes?
- Kubernetes Clusters
  - Nodes
- Kubernetes Components
  - Pods
  - Deployments
  - Services
- Helm
  - What is Helm and why use it?
  - Charts
    - Chart Structure
    - Chart Dependencies
  - Repositories
  - Releases

5.2.2 Useful Concepts

You should have a general familiarity with the following concepts, but deep understanding is not necessary.

- Services
- Storage
  - Volumes
  - Persistent Volumes
  - Persistent Volume Claims
- Configuration
  - ConfigMaps

5.2.3 Practical Skills

You must be able to do the following:

- Kubernetes Basics:
  - Deploy a containerized application on a cluster.
  - Update the containerized application with a new software version.
  - Debug the containerized application.
- Basic Kubectl Commands
  - Get info about nodes, pods, deployments, services
  - Create and delete resources
  - Get logs
  - Open an interactive shell on a running pod for debugging
- Helm Basics
  - Read simple Helm Charts
  - Create a simple Helm Chart from scratch
  - Deploy a Helm Chart

CHAPTER

SIX

PYTHON

You will need a solid grasp of the fundamentals of Python before starting Dev School. We will work through more advanced features of the language but expect you to be able to code to a basic level already. If you can complete the attached exercise then you will be at a comfortable level to begin the school.

6.1 Resources

6.1.1 Recommended Exercises

Write code! The best way to get better at a programming language is by writing code.

One way to do this is through Code Katas. There are many lists of Code Katas online. Here is one:

- Code Katas

6.1.2 Recommended Reading Material

- Python 3 Tutorial
- Alternative Tutorial

6.2 Exercise

Exercise

6.3 Checklist

6.3.1 Python Language Basics

You should be familiar with the syntax for programming in Python, especially following:

- Essential Syntax and Fundamentals
  - Defining variables and functions
  - Defining classes and inheritance
    - Operator overloading
  - Type hinting (especially if you have not worked with a statically typed language)

  - Operators, control flow
  - `range` and `enumerate`
  - `iter` and `next`
  - Standard library data structures (lists, dicts, sets, tuples) and unpacking syntax
  - f-strings
  - Comprehensions
- Useful items
  - Decorators
  - Dataclasses
  - Context Managers
  - File handling
  - Modules and packages
- Optional
  - Multithreading and multiprocessing
  - Higher order functions (eg passing a function to `map` a collection)
    - lambda functions
    - closures
  - Generators
  - Async python
  - 'Modern' python (3.11) features such as `match case`
- Environments
  - Setting up a `venv` or a conda `env`
  - Jupyter notebooks (try installing and running one locally)

Python & Pandas Fundamentals Exercise

Bank Transactions Simulator

For this exercise, you will build a bank transactions simulator. Users can create accounts, make deposits, withdrawals, and view transaction history. While you should practice building a small project, it may be useful to experiment with the code in a jupyter notebook first. You may use google / ChatGPT / any other resources to help if a concept is unfamiliar for you.

Python Basics

1. Create some datamodels to model the bank accounts and transactions, using `@dataclass`. You will need
    - Account (Needs to have an `account_holder`, `account_number`, `balance` and a list of `transactions` (see next line). What datatypes are best to represent each field)?
    - Transaction (these will populate the list held in each account). What types of transaction are there? What fields are needed to represent them and can any be optional?
    - e.g. think about transfers, which move money to or from another account, and deposits/withdrawals.
2. Create `CurrentAccount` and `SavingsAccount` classes that inherit from the `Account` class with additional properties if needed. (e.g. daily limits)
3. Overload the "less than" operator for `Account` class to compare the balance of two accounts.
    - Hint: which dunder methods are used for comparison operators?
4. What is printed when you pass an `account` object to `print`? Add an override to the Account class so that this instead prints a nicely formatted account statement.
    - Hint: which dunder method defines how an object is converted to a string?
    - Multiline f-strings
5. Implement a `Bank` class which holds accounts.
    - What would be a good datastructure use to model this? Consider which fields of an `account` are guaranteed to be unique for each account
    - Pull account(s) information for a given `account_holder`.
    - Add a method which create new accounts for a list of new `account_holders`. First implement this with a loop, then try changing it to a comprehension.
    - Add methods to deposit/withdraw money into an account.
    - Add a method which can transfer money from one account to another, creating the relevant `Transactions` and updating the accounts.
    - What should happen when someone tries to transfer or withdraw more money than they have?
6. When a transaction occurs, write the transaction to a csv file.
    - This should be a single new file each time you create the `Bank`. Try using the `datetime` library to put the current time in the filename.
    - You can use the `open` keyword as a context manager
    - Consider what columns the csv file should have and write these as the first line, when the `Bank` is initialised.
    - Append each transaction to this file as they occur.
    - Hint: You can use `",".join(...)` to construct the line.
7. Write a script which can
    - Generate 100 account names (random strings are fine) and initialise the bank with an account for each.
    - Deposit some money into each of them,
    - Simulate 1000 random transactions between accounts. Choose two accounts at random and pick a random amount. What bounds might make this sensible?
    - Check your log file is the expected length. Make sure you create a new log file each time you run the simulation!

8. The bank wants to find inactive accounts. Use `filter` with a lambda function to find accounts with less than 10 transactions.

Pandas

1. Set up your project within a virtual environment, and `pip install pandas`
2. After running the simulation and generating the CSV file with transaction data, perform the following tasks in a notebook, using pandas:
3. Load your CSV file of transactions into a DataFrame.
4. Use `group_by` and `concat` to construct a new dataframe with `account_number` as the index and the following columns
    - number of transactions
    - current balance
    - `min_balance` and `max_balance` over the whole simulation
5. Reproduce the list of `inactive_accounts` from the first part, using Pandas operations on the transactions dataframe. Is the list of transactions enough information to guarantee we find them all?
6. If your bank implementation allows negative balance, find out which accounts went into overdraft at any point. If not, check if anyone's balance dipped below a threshold.
7. Find which account made the most transactions and plot the balance of this account over time.

CHAPTER

SEVEN

PYCHARM

An IDE is essential to improving your productivity as a Software Engineer. PyCharm is the preferred Python IDE for the vast majority of Python teams at IMC, and the IDE with the most internal support.

7.1 Recommended Reading

- PyCharm - Getting Started

7.2 Recommended Viewing

- Youtube - Be More Productive With IntelliJ IDEA
- Youtube - IntelliJ IDEA Debugger Essentials

7.3 Practical skills

Be able to use the following features:

- Reading Code
  - Code Navigation
  - Find Usages
- Writing Code
  - Inspections
  - Live Templates
  - Local History
  - Code Style and Formatting
- Refactoring
  - Renaming
  - Change Signature
  - Extract/Introduce refactorings
  - Inline

  - Safe Delete
  - Migrate
- Running
  - Running a Python application
  - Running tests
- Debugging
- Venv Integration