First Year Overview

101 Programming Fundamentals

This module introduces programming constructs that are the foundation of any kind of programming. For many students, this is their first time programming. To overcome the technical vocabulary of computer science, we present the fundamentals of programming in plain language. Once the conceptual understanding is embedded, then only is the computer science terminology introduced.
In this module, the language of instruction used is Python. We chose Python because it is one of the more accessible programming languages to understand and is widespread globally.

By the end of this module, students:

- Will know the programming fundamentals, including variables, branching, loops, data structures, procedures and functions, error and exception handling, packaging their code into modules and using external libraries;
- Are comfortable using unit tests to prove their programs' functionality and write elementary unit tests of their own; have practised the basics of git and branching;
- Will have learned and implemented OAuth using the Google OAuth API; have written code that integrates with an external HTTP REST API, including JSON;
- Can write programs that use configuration files to control the parameters of a system; and
- Know how to import data stored in text files

WeThinkCode_
info@wethinkcode.co.za
www.wethinkcode.co.za

Programming

- **Making Decisions:** Make a program do different things based on the data it receives.
- **Repeating Instructions:** Get a program to do the same thing several times over.
- **Structuring Data:** Combine data into meaningful structures.
- **Combining Instructions:** Combine several instructions and reuse the combined instructions as a single instruction.
- **Processing Collections of Data:** Work with multiple instances of the same kind of data in various ways.
- **Modules and Packages:** Use code from other developers (including open source) to construct a more extensive program from smaller modules.

Engineering

- How to structure code files and package them to be run.
- How to write a program and test that the core program is functioning as intended.
- How to run automated tests
- Introduction to Test-Driven Development

Communication

- **Journaling:** Reflecting on various learnings and their application and summarising them, in written format, in a journal. Journaling hones the ability to describe technical concepts and summarise them in a way that others can understand. The more reflective a developer is, the better they are at communicating.

WeThinkCode_
info@wethinkcode.co.za
www.wethinkcode.co.za

Presentations: Delivering short-form presentations known as Pecha Kucha to peers. Giving presentations develops public speaking skills and the ability to prepare slides and use them to deliver information effectively

Team Work

- **Collaboration:** Working in small teams to build a program that, ordinarily, is too much for one person to complete on their own in the prescribed time. Students must collaborate in groups of 4 to build a program that works end-to-end.
- **An Iterative Way of Working:** The teams work on a group project in 3 iterations of 2 weeks each. Time-boxing the scope of work introduces concepts of planning and the necessity of shared understanding.
- **Demos:** At the end of the semester, each group present their projects to staff and guests from the industry.
- **The Team Project:** The brief is to build a console-based system that manages the Code Clinics. The project requires students to use their fundamental knowledge to integrate with the Google Calendar API to implement the code clinic schedules. Each team collaborates on the code using a shared git repository on GitHub.

102 Object-Oriented Programming

This module is an introduction to software design through the lens of Object-Oriented Programming (OOP). We assume that students are comfortable with fundamental programming constructs introduced in the 101 Programming Fundamentals module. In this module, we introduce Java as the programming language. The switch from Python to Java further entrenches the fundamentals of programming

By the end of this module, students:

- Will know the basics of object-oriented programming such as encapsulation, inheritance, polymorphism and composition;
- Would have refactored procedural code to object-oriented code; have been introduced to DevOps by using GitHub actions to build, test and package
- their programs;
- Have programmed a client-server architecture system from scratch;
- Will know the fundamentals of network programming by programming with sockets;
- Have written a multi-user system using thread-based concurrency; and
- Will learn how to implement a custom application protocol with JSON

Programming

- **Java Fundamentals:** How to use the basic programming constructs in Java to mould the design of a program.
- **Encapsulation:** Create classes to implement behaviour, hide and govern access to data.
- **Inheritance:** Use abstraction as an instrument of design.
- **Polymorphism:** Implement specialisation of objects at run-time, based on the abstraction designed.
- **Composition:** Combine objects to create more complex objects.

Engineering

- **Build tools:** Build using Apache Maven, the most popular build tool for Java programs, to manage program dependencies and compile and test programs.
- **Unit Testing:** Use JUnit to reinforce existing testing techniques.
- **Refactoring:** Refactor a procedural program to an object-oriented program.

WeThinkCode_
info@wethinkcode.co.za
www.wethinkcode.co.za

Communication

- **Team Leads & Daily practices:** Team leads are nominated to be the interface to each team. The teams continue honing their agile techniques such as stand-ups, managing a task board, iteration planning, showcases and retrospectives.

Team Work

- Students must collaborate in teams of 4 to build a program that works end-to-end. The project consists of 3 two week iterations.
- Students follow agile practices for building software, including pair programming, daily standups, showcases, and retrospectives at the end of each iteration. A final demo takes place at the close of the semester.
- **The Team Project:** The brief is to build a multi-user network-based game called RobotWorlds wherein players launch robots into a grid world and combat each other. The system requires students to apply their knowledge of object orientation to implement client socket-based communication over a network and a multi-threaded server. The choice of the user interface is left to the team to decide. Groups share their code on GitHub and are required to use GitHub actions to automate their programs' build, test, and packaging.

Second Year

201 Brownfields Software Development

Programming

Automated Testing: Write unit tests, acceptance tests and integration tests.
Persistence: Use a relational database and SQL to store and retrieve data.
Mobile Development: Develop a mobile application against an existing back-end system.
Code Analysis: Analyse a codebase to establish its technical debt and code quality.
Refactoring: Refactor existing code to improve its quality and design by applying TDD.
Design Patterns: Apply patterns to enhance the structure at the code level.
Fundamentals of Software Architecture: Understand coupling and cohesion.

Engineering

- **DevOps:** Create build pipelines for continuous integration and continuous delivery, including automated testing and deployment with Docker.
- **Effective Version Control:** Use trunk-based development techniques to maintain a stable main branch of the codebase.
- **Secrets Management:** Use PGP to store and access credentials used in production environments securely.

WeThinkCode_
info@wethinkcode.co.za
www.wethinkcode.co.za

Communication

- **Showcases:** Every iteration ends with a showcase. Students prepare and present their work for that iteration to mentors and staff.
- **Technical Documentation:** Illustrate the architecture and design of the system under development, from component level down to modules and classes.

Team Work

- **Planning:** Students are given goals for each iteration and plan, estimate and prioritise their work to achieve the iteration goals.
- **Taskboard:** Use a task board to manage the backlog of work, track work progress, and keep the board up to date via daily stand-ups.
- **Retrospectives:** Reflect on the iteration and identify and implement improvements in the way the team works.
- **Pairing:** Practice pair programming to increase shared understanding and quality of the code.

202 Distributed Systems Architecture

We continue with team-based software development and focus on the architecture for distributed systems and their implementation on cloud platforms.

By the end of this module, students:

- Will have worked in a multi-team environment using agile principles and practices;
- Understand and implement the fundamentals of distributed systems architecture;

- Will have built a system with cloud technologies;
- Will have written secure code based on the OWASP top 10 code vulnerabilities;
- Will have programmatically provisioned the infrastructure for a cloud-based system;
- Will have written a web application;
- Will be familiar with asynchronous programming;
- Understand and can apply Domain Driven Design patterns and techniques;
- Will have written code for load and stress tests; and
- Are familiar with the design of an API.

Programming

- **Decomposition:** Design discrete, single responsibility services that collaborate with other services in a distributed system.
- **Security:** Write code that protects against common security vulnerabilities.
- **API Design:** Design and implement an API for services.
- **Web development:** Build web applications that use multiple API's.
- **Asynchronous programming:** Write code that executes in a predominantly asynchronous fashion; and the automated testing thereof.
- **Domain-Driven Design:** Apply DDD patterns at the code level and the systems level.

Engineering

- **Infrastructure as Code:** Programmatically provision the infrastructure for the system under development, being a critical DevOps technique.
- **Load Testing:** Create and execute tests that stress the load and performance of a system.
- **Cloud-Native:** Design and deploy a distributed system on a cloud platform.

WeThinkCode_
info@wethinkcode.co.za
www.wethinkcode.co.za

Communication

- **Business Analysis:** Learn and apply effective business analysis techniques to communicate requirements precisely.
- **Architecture:** Create and present architecture slide decks for the system under development.
- **Technical Documentation:** Document the API for use by other developers.

Team Work

- **Continuous Flow:** Students transition from time-boxed iterations to practising continuous delivery.
- **Multi-team collaboration:** Manage dependencies between teams and communicate between teams.