COURSE DURATION

16 hours

COURSE SYNOPSIS

This course sets out essential concepts and skills relating to the ability to use computational thinking and coding to create simple computer programs.

COURSE OBJECTIVES

After the course, you will be able to:

- Understand key concepts relating to computing and the typical activities involved in creating a program.
- Understand and use computational thinking techniques like problem decomposition, pattern recognition, abstraction and algorithms to analyse a problem and develop solutions.
- Write, test and modify algorithms for a program using flowcharts and pseudocode.
- Understand key principles and terms associated with coding and the importance of well-structured and documented code.
- Understand and use programming constructs like variables, data types, and logic in a program.
- Improve efficiency and functionality by using iteration, conditional statements, procedures and functions, as well as events and commands in a program.
- Test and debug a program and ensure it meets requirements before release.

TRAINING DELIVERY METHODOLOGY

Classroom training with hands on (using the computer) is one of the best ways to attain a grasp of a new programming language

TARGET AUDIENCE

Data scientists, finance/accounting professionals, consultants, project managers, agile / scrum professionals, web developers, administrators and other professions who require some basic knowledge of python.

ASSUMED SKILLS

- Learners must be able to read, write, speak and listen to English at secondary school level
- Learners must be able to operate computers at intermediate level

COURSE OUTLINE

1 Computing Terms

1.1 Key Concepts

- Define the term computing.
- Define the term computational thinking.
- Define the term program.
- Define the term code. Distinguish between source code, machine code.
- Understand the terms program description

and specification.

- Recognise typical activities in the creation of a program: analysis, design, programming, testing, enhancement.
- Understand the difference between a formal language and a natural language.

2 Computational Thinking Methods

2.1 Problem Analysis

- Outline the typical methods used in computational thinking: decomposition, pattern recognition, abstraction, algorithms.
- Use problem decomposition to break down data, processes, or a complex problem into smaller parts.
- Identify patterns among small, decomposed problems.
- Use abstraction to filter out unnecessary details when analysing a problem.
- Understand how algorithms are used in computational thinking.

2.2 Algorithms

- Define the programming construct term sequence. Outline the purpose of sequencing when designing algorithms.
- Recognise possible methods for problem representation like: flowcharts, pseudocode.
- Recognise flowchart symbols like: start/stop, process, decision, input/output, connector, arrow.
- Outline the sequence of operations represented by a flowchart, pseudocode.
- Write an accurate algorithm based on a description using a technique like: flowchart, pseudocode.
- Fix errors in an algorithm like: missing program element, incorrect sequence, incorrect decision outcome.

3 Starting to Code

3.1 Getting Started

- Describe the characteristics of well-structured and documented code like: indentation, appropriate comments, descriptive naming.
- Use simple arithmetic operators to perform calculations in a program: +, -, /, *.
- Understand the precedence of operators and the order of evaluation in complex expressions. Understand how to use parenthesis to structure complex expressions.
- Understand the term parameter. Outline the purpose of parameters in a program.
- Define the programming construct term comment. Outline the purpose of a comment in a program.
- Use comments in a program.

3.2 Variables and Data Types

- Define the programming construct term variable. Outline the purpose of a variable in a program.
- Define and initialise a variable.
- Assign a value to a variable.
- Use appropriately named variables in a program for calculations, storing values.
- Use data types in a program: string, character, integer, float, Boolean.
- Use an aggregate data type in a program like: array, list, tuple.
- Use data input from a user in a program.
- Use data output to a screen in a program.

4 Building using Code

4.1 Logic

- Define the programming construct term logic test. Outline the purpose of a logic test in a program.
- Recognise types of Boolean logic expressions to generate a true or false value like: =, >, <, >=, <=, <>, !=, ==, AND, OR, NOT.
- Use Boolean logic expressions in a program.

4.2 Iteration

- Define the programming construct term loop. Outline the purpose and benefit of looping in a program.
- Recognise types of loops used for iteration: for, while, repeat.
- Use iteration (looping) in a program like: for, while, repeat.
- Understand the term infinite loop.
- Understand the term recursion.

4.3 Conditionality

- Define the programming construct term conditional statement. Outline the purpose of conditional statements in a program.
- Use IF…THEN...ELSE conditional statements in a program.

4.4 Procedures and Functions

- Understand the term procedure. Outline the purpose of a procedure in a program.
- Write and name a procedure in a program.
- Understand the term function. Outline the purpose of a function in a program.
- Write and name a function in a program.

4.5 Events and Commands

- Understand the term event. Outline the purpose of an event in a program.
- Use event handlers like: mouse click, keyboard input, button click, timer.
- Use available generic libraries like: math, random, time.

5 Test, Debug and Release

5.1 Run, Test and Debug

- Understand the benefits of testing and debugging a program to resolve errors.
- Understand types of errors in a program like: syntax, logic.
- Run a program.
- Identify and fix a syntax error in a program like: incorrect spelling, missing punctuation.
- Identify and fix a logic error in a program like: incorrect Boolean expression, incorrect data type.

5.2 Release

- Check your program against the requirements of the initial description.
- Describe the completed program, communicating purpose and value.
- Identify enhancements, improvements to the program that may meet additional, related needs.