higher education & training

Department:
Higher Education and Training
REPUBLIC OF SOUTH AFRICA

NATIONAL CERTIFICATES (VOCATIONAL)

SUBJECT AND ASSESSMENT GUIDELINES

Programming

NQF LEVEL 2-4

IMPLEMENTATION DATE: JANUARY 2023-2025

NCV Programming L2 – L4

Also referred to as Technical Programming

Contents

NCV CS/IT Programme

	Level 2 – Basic principles of Computer programming and Computer literacy	Level 3 – Introduction to Technical Programming	Level 4 – Technical Programming
Fundamentals* *Note: The 3 fundamental subjects are compulsory The 3 core vocational subjects are also compulsory	• English/Afrikaans/IsiXhosa (First Additional language) • Life Orientation • Mathematics **OR** Mathematical Literacy	• English/Afrikaans/IsiXhosa (First Additional language) • Life Orientation • Mathematics **OR** Mathematical Literacy	• English/Afrikaans/IsiXhosa (First Additional language) • Life Orientation • Mathematics **OR** Mathematical Literacy
Information Technology & Computer Science With specialization in programming and robotics	• Electronics and Digital Concepts for Robotics	• Electronics and Digital Concepts for Robotics	• Electronics and Digital Concepts for Robotics
	• Robotics Fundamentals	• Introduction to Robotics	• Robotics and Industrial automation
	• Basic principles of Computer programming and Computer literacy	• Introduction to Technical Programming	• Technical Programming
	• Physical Science (O)* ***OR***	• Physical Science (O)* ***OR***	• Physical Science (O)* ***OR***
	• Instrumentation Technology (O)* ***OR***	• Instrumentation Technology (O)* ***OR***	• Instrumentation Technology (O)* ***OR***
	• Mechatronic Systems (O)* ***OR***	• Mechatronic Systems (O)* ***OR***	• Mechatronic Systems (O)* ***OR***
	• Engineering Graphics and Technology (O)* ***OR***	• Engineering Graphics and Design (O)* ***OR***	• Mechanical Draughting and Technology (O)* ***OR***
	• Electrotechnology (O)*	• Electro-technology (O)*	• Electro-technology (O)*

Key

Symbol	Description
	Existing NCV Subject
	New subject
	Elective (O*)

Overview of the subject

The digital age we currently find ourselves in resulted from the third industrial revolution in communications and information technology. The next phase in industrial development is the integration of production, sustainability, and customer satisfaction. Technologies widely used to facilitate this integration includes robotics, artificial intelligence, 3D printing, biotechnology, cloud computing, and big data. With the end result being a number of hi-tech industrial processes that characterize the 4th industrial revolution (Bloem et al. 2014; Ellen Frederick 2016).
In the past, it was acceptable for a worker to "only operate" later workers were required to "understand and operate" with the current technologies part of the new hi-tech industrial processes that a worker is confronted with the new mantra is "operate understand and modify".
To not only work with a piece of technology but actively modify the operating parameters to suit the current and future industrial needs will require a worker to speak the language of technology.
Workers will need to program robots and 3D printers, programmatically interact with biotechnology, train or verify artificial intelligence against big data or logically connect and orchestrate the different internet of things devices connected to the cloud.

The aim of this subject is to introduce programming as a form of communication and expression to accomplish the above providing the student with a solid foundation in logical thinking and enabling them to design and present stepwise solutions and transform the abstract into the symbolic.
It also incorporates the study of the integrated components of a computer system (hardware and software) and the practical techniques for their efficient use and application to solve everyday problems.

The solutions to problems are designed, managed and processed via end-user applications, software development tools (including visual programming tools) and communicated using appropriate information and communication technologies (ICTs)
The subject content includes knowledge (described as theoretical and or factual knowledge) and skills described as:

1) Cognitive: involving the use of logical, intuitive, creative and computational thinking,
2) Practical: involving the use of steps, procedures, principles, techniques, strategies and tools.

Figure 3 presents the cognitive complexities involved, which refer to the technological skills as the bases, communication and information management as higher skills and programming skills as the highest level.
Programming is somewhat unique as it requires both knowledge (theoretical and or factual) and skills (cognitive and practical). Because of this the teaching of programming needs a strong Pedagogical Content Knowledge (PCK). PCK as defined by Shulman (1986) is: “The ways of representing and formulating the subject that make it comprehensible to others.” Saeli et al. (2011)

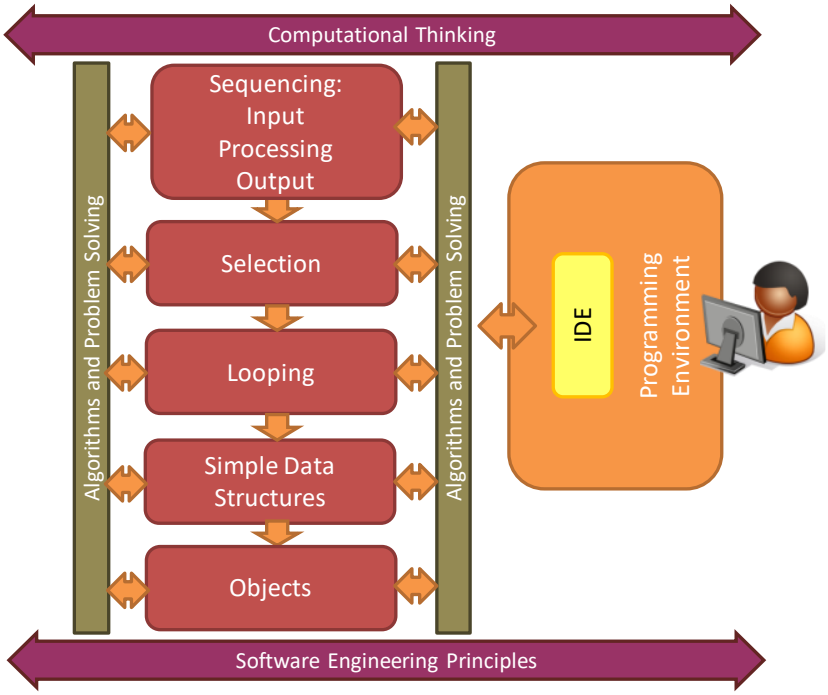
attempted to define the PCK's required when teaching programming by answering four interconnected questions:

1. Why teach programming?
2. Programming and the world of robotics
3. What should be taught?
4. What are the teaching difficulties?
5. How should the topic be taught?

A clear answer to the above four questions will not only guide the subject presenter in teaching but will also motivate the student learning as it sets a clear purpose, vision and mission for NCV Programming L2 – L4.

Why teach programming?

Students value immensely from learning programming. Because programming requires the student to generate a solution to a problem, they acquire new problem solving, design and critical thinking abilities. They learn the value of breaking a large problem into smaller sub problems and reusing previous solutions in new problems. Identifying, solving and communicating solutions to problems using a structured programming language syntax also contributes to the students' natural language skills, because "they are required to learn to tell, in an un- ambiguously way, what they want the computer – an unintelligent machine – to perform". (Hromokovi˘c, 2006).
The skills acquired from studying programming as also been found to be transferable to other subjects. Students will indirectly learn and associate programming knowledge with mathematical concepts like rigorous thinking, variable, function, de- composition, debugging and generalization (Syslo & Kwiatkowska, 2006).



Because programming is a new generation subject, it brings together pieces from different areas such as: science, mathematics and the art of intellectual model building (Papert, 1980; Mulder, 2002). This completeness gives students the opportunity to be faced with a multi-disciplinary subject that connects different aspects in a single class.

Programming and the world of robotics

The subject name Technical Programming leans towards the design and creation of programs that amongst others enable the physical computing domain. Programming and incorporating robotics strengthen critical thinking, develops an advanced logical and computational ability, and makes students far better at teamwork and analysis than others. This subject on its own provides a student with the opportunity to explore and engage with the software development process by writing programs to solve problems in real live contexts. Typically, computer programming involves creating an executable computer program that performs a particular computation. It includes tasks such as analysing information, generating algorithms and evaluating programs accuracy.

What should be taught?

Learning programming requires two kinds of knowledge, program generation and the program comprehension (Mannila, 2007). Program generation requires a problem to be analysed, an algorithm be produced, and then translated into a program code. Program generation will thus not just include knowing the syntax of the specific programming language but also the process of problem solving, reflection on this process, and in the development of algorithmic ways of thinking. (Breed et al., 2005; Ginat, 2006). For program comprehension, the programmer is required to analyse existing programs in order to demonstrate understanding of how the program works and how the program will respond with output based on input provided.

Although specific aspects will differ based on the specific programming language implementations Govender (2006) identified three main aspects students need to learn irrespective the programming language used: data, instructions and syntax. Data refers to the concepts of variables and data types, instructions requires the understanding on control structures and subroutines and syntax is the group of rules that determine what is allowed and what is not within a programming language.

What are the teaching difficulties?

Students will experience several difficulties when first learning programming. First among them as identified by DuBoulay (1989) is students finding it difficult to rationalize the usefulness and benefits of learning how to program. Others include understanding the general properties of the machine that one is learning to control, formal language syntax and semantics, how to structure and logically order or combine small-scale program units or concepts (e.g., using a loop) to achieve a solution and learning the skill to specify, develop, test and debug a program using the available tools.

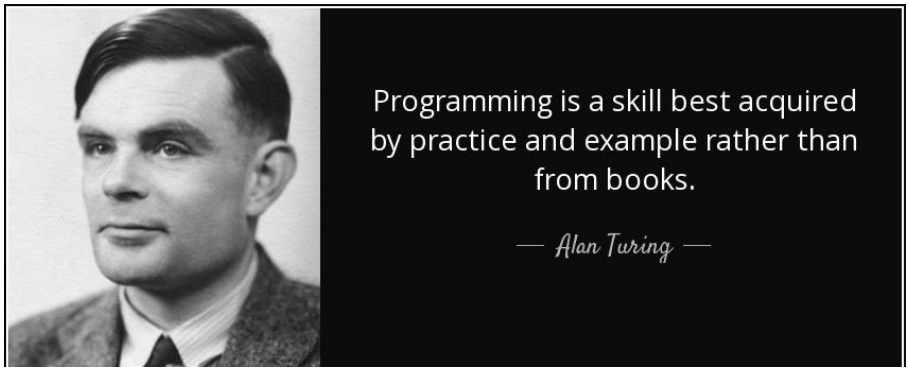
How should the topic be taught?

Students will have different learning needs, thus there is no one correct approach when teaching programming. What may be beneficial is to incorporate several different approaches and methods as the circumstance requires. The first approach is historical development recommended by Abelson and Sussman (1996) where students will start writing simple solutions and then combine those simple solutions into more complicated programs. This allows for gradual cognitive development and has the additional benefit of teaching students the concept of modularity.

To achieve algorithmic thinking, it has been shown that students should be encouraged to solve as many programming language independent problems as possible. Both Bell, Witten and Fellows (1998) and Curzon and McOwan (2008) have shown that algorithmic thinking can be introduced without the need (and added complexity) of a computer. An example of this is using a paper-based IPO chart to indicate what input is needed for the steps to solve a problem and what the resulting output will be.

It is also recommended to provide students with a framework of understanding a model or description of the internal working of the machine they are programming should be provided. This model can have a theoretical base in computer science but should be simple to interpret, be relatable and should aid students in understanding how a computer uses the programming langue instructions to store and process data.

Finally programming concepts should be introduced to student in an engaging way. Several programming environments have been designed to introduce programming in a visual environment where the focus is not on syntax but rather the semantic and problem-solving aspects of programming (Resnick et al., 2009). One such visual programming environment is Scratch. Numerous research projects (https://scratch.mit.edu/info/research/) suggest the value of Scratch to teach coding to novices, e.g. Meerbaum-Salant, et al. (2010) found that a programming course that used Scratch, resulted in meaningful learning. Students were able to internalise programming concepts and showed improved cognitive achievement.



Main topical areas

Topic 1 Computer hardware and software

This topic will expose the student to hardware and software fundamentals of a computer. The goal is to provide the student with a thorough understanding of the underlying hardware that eventual applications will run on as managed by the underlying software.

Topic 2 Problem solving in computer programming

Problem solving is a key component to writing software applications. This topic equips the student with language agnostic problem-solving techniques and tools. The topic also introduces the student to the concept of an algorithm as a generic set of steps to solve a problem.

The solutions created in the form of algorithms are then transferred and recoded to a block-based language to illustrate the correctness of the algorithmic solution.

Topic 3 Concepts of programming for single board microprocessor or microcontrollers

Tightly coupled to topic 2, topic 3 covers a visual programming approach to solving problems. The topic also introduces the student to the concept of computer programming and provides a forgiving environment and gentle introduction to solving problems using algorithms.

This topic emphasises the use of a visual programming language to present the students with a foundation towards mastering the programming constructs of the high-level programming language used in this curriculum.

Various block-based visual programming languages are available to assist students with mastering programming, computational thinking, and coding related problems. This includes the use of block-based coding tools to facilitate GPIO functionality and processing on single board computers.

This topic is also developed bearing in mind topic 5 of the robotics curriculum and is integrated as such.

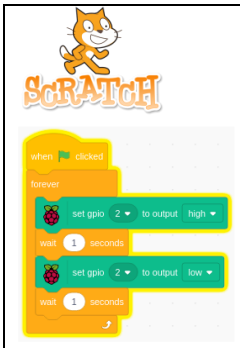
- In Level 2 the emphasis is placed on Scratch which in conjunction with the Raspberry Pi allow for GPIO programming.
- In Level 3 the students are further introduced to block-based programming for Arduino and the Raspberry Pi.
- For Level 4 the students' knowledge base is further expanded with the introduction of Python using the Arduino IDE.

Topic 4 Programming tools and utilities

This topic introduces the essential tools and utilities required to write applications. Installation and configuration are covered, and essential terms are defined. The topic then moves beyond simply compiling and running applications to cover the concept of debugging a running application in real time. The topic also introduces students to the tools and concepts involved when working with a team on a software project.

Topic 5 to 12 (Solution Development)

Programming Languages

 **Level 2** **GPIO & Arduino block-based programming Level 3**	Various studies suggest and motivates the benefits of block-based visual programming to expose novices to the concept of programming, coding, and computational thinking (see, Asgedom, 2021 & Kong, & Wang, 2021). Scratch was created by Massachusetts Institute of Technology (MIT) and designed especially for ages 8 to 16. Scratch has a huge community with numerous resources that includes learning guides, teaching guides, lessons, videos, etc. Scratch offers a low floor (easy to get started, even with no experience of coding), wide walls (allows for all kinds of projects – from animations and games to science experiments) and high ceiling (the possibility to create complex projects) with a playful experience that teaches novices to code in an experimental fashion – it starts where the beginner is and allows for incremental development. (https://scratch.mit.edu/info/research/) suggest the value of Scratch to teach coding to novices, e.g. Meerbaum et al. (2010) found that a programming course that used Scratch, resulted in meaningful learning. Students were able to internalise programming concepts and showed improved cognitive achievement. Scratch shipped with the Raspberry PI allows for the creation of code that enable GPIO programming on a concrete level. It also enables the student to be exposed to physical computing. Please see (https://projects.raspberrypi.org/en/projects/physical-computing-with-scratch/3)
C/C++ **Levels 3 and 4** **In Level 4 (as required per project)**	Udacity.com (2021) states that: C and C++ remain two of the most popular programming languages in use today. These languages have laid the foundation for many other languages and are great options for starting your coding journey. With regard to robotics programming various practitioners agree that

	C++ and Python are probably the 2 most used languages when it comes to program robots. C++ for performance and Python for the ease of use (Roboticsbackend.com 2021) The University of Sheffield as cited by Futurelearn.com (2021) explains that: Programming for the Arduino microcontroller is based on the C/C++ language, one of the most widely used programming languages in robotics. This is an ideal way to learn the basics of this important language while using robotics hands-on.
 **Levels 2 and 4**	Python ranks among the most popular and fastest-growing languages in the world. Python is a powerful, flexible, and easy-to-use language. In addition, the community is very active there. It is used in many organizations as it supports multiple programming paradigms (Mehla, 2021) There are several Python-based packages for robotics, each reflecting different design approaches or requirements and with various levels of finish in terms of documentation, examples, and continuous integration (Corke & Haviland, 2021) Python's strengths as a programming language for robotics programming are: - Easy to learn and easy to read. - Gives access to many powerful libraries. - Quick to write usable (and reusable) code. - It's very popular so there's lots of help available from the Python community (Owen-Hill, 2018)

LTSM Development and teaching strategy

The developers of any LTSM should consider the fact that a curriculum statement and syllabus, such as the one provided here, is made up of various contextual topics and subject outcomes and should therefore not be used as a table of contents for writing textbooks.

The topic and subsequent subject and learning outcomes' numbering also does not necessarily indicate the sequence in which material should be presented.

Vised and knowledgeable subject experts and LTSM developers will logically group different subject and learning outcomes and will present these in an educationally sound, coherent, and sequenced manner to provide students with an authentic integrated learning experience.

Core competencies and skills should be presented covering both the theoretical and practical aspects of the subject.

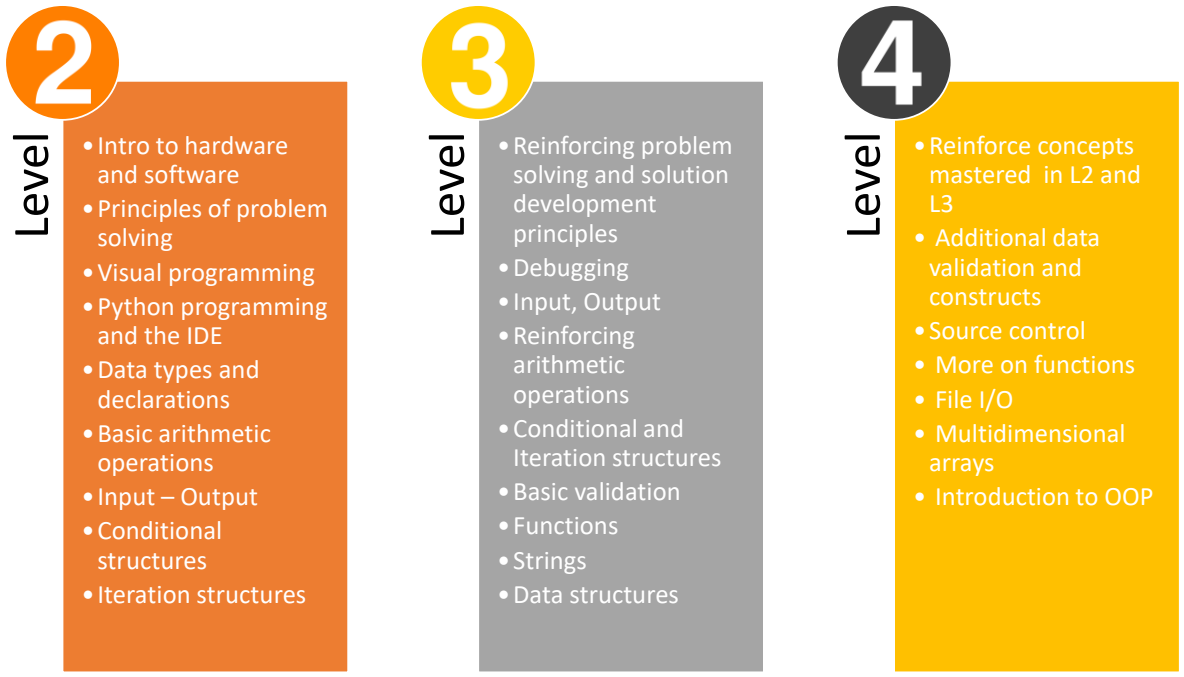
Lecturers should design their own work schedules (or use/adapt the work schedule provided in their textbook) to appropriately group and sequence content considering comfortable pacing of the curriculum matter.

The SOs and LOs presented should not be seen as stand-alone topics but relevant SOs and LOs or content should be presented in an integrated manner.

Integration of curriculum content in the lesson presentation should flow naturally according to the nature, links, and 'overlap' of the content. For example, some content from one SO and LO may strengthen and underpin the content of another. This approach should be applied throughout the three-year curriculum.

It is important that the specific technologies in the teaching plans are revised at regular intervals to phase out old technologies and to include new technologies.

Snapshot of the curriculum



Lecturer profile

The academic and qualifications profile of a typical lecturer is an individual:

- With an NQF Level 6 qualification (minimum NQF Level 7 preferred) in Computer Science, Information Technology, Computer System Engineering, Informatics, or other Software Development related discipline.
- Who has proficient skills in:
  - communication, listening, collaboration, adaptability, empathy, and patience, which stimulates an engaging classroom presence, ignites and values problem solving, values real-world learning, logical thinking, exchange and transfer of best practices and, has a passion for lifelong learning.
  - Logical reasoning and creativity as well as the ability to identify and solve complex and real-life problems.

Teaching time and offering type

The instructional offering is presented full time over two semesters **spanning over 24 weeks** (**not including assessment**)

This is equivalent to:

- 24 x (2 x double periods of +/- 1 hour 50 minutes) + 1 (single period of +/- 50 minutes)
- 56 hours (+/- 4.5 hours per week) of which -
  - 36 hours are instructor led contact hours (which also include practical's sessions),
  - 18 hours are instructor led contact hours dedicated to practical tutorials (on PCs) and
- In addition, a further 10 hours per term, (i.e. +/- 1.5 hours per week) are required for students to practice on their own to reinforce and master practical concepts and skills. (This should be accommodated in open labs provided by the college)

Resources

In Programming, students are required to work individually on a computer during contact time and need access to the Internet.

Management plan, infrastructure, equipment and finance

Colleges must have a management plan for the subject that addresses the following:

- Initial capital layout for setting up a computer laboratory. The financial plan should provide for the following:
    - Mid-range computers (to ensure a lifespan of 4 - 5 years), networked
    - One computer / workstation per student per period (during contact time)
    - Provision for sufficient computers to enable the assessment to be completed in one or two sittings
    - Internet access
    - One high-speed printer per network
    - Data projector per lab and classroom
    - Classroom management software

Physical resources

- Lecture room(s)
- Training area-work area
- Ablution facilities
- NB: Safe storeroom to be equipped with enough storage space for consumables suitable for storage.
- Funds, from learning provider or funding bodies, for the procurement of consumables, tools and equipment need to be made available for the effective operation of a workplace involved in a training programme. Students need to be individually equipped with the necessary tools.

Learning and teaching materials

Learning materials must conform to approved training and industrial standard requirements and articulate to Higher Education. Learning support materials and resources are needed for both academic and practical aspects of learning, and include the following:

- Comprehensive texts for student use as well as for supplementary reading.
- Various robotic kits and electronic equipment
- Learning materials for projection during lectures.
- Facilities to support the promotion of opportunities for research by both students and lecturers.
- Educational tours to relevant learning venues.
- Educational and motivational talks from industry.
- Visual and audio-visual material
- Relevant workshop manuals and documentation.
- Models and demonstrations.

Software Requirements

- Operating system
- Security software – antivirus programs
- Web browser e.g. Microsoft Internet Explorer, Microsoft Edge, Chrome or Mozilla Firefox
- Multimedia software, i.e. Adobe Flash Player (latest versions – free online installation)
- PDF Readers
- Compression and extraction software (E.g. Zip, 7Zip)
- C/C++ compiler and tools

Budget

- Annual running costs
    - Software licensing (operating system, application software, security software)
    - Cartridges, paper, storage media
    - Breakages and maintenance (regular service plan)
    - Insurance
    - Internet connectivity
- Sustainability plan
    - To upgrade or replace software and equipment every 4 - 5 years.
    - Safety standards in terms of electrical points and Distribution boards (DB) must be met. Surge arrestors should be installed. Electrical work within the computer lab must carry a certificate of compliance from an appropriate service provider /authority.
    - All anti-theft and fire safety measures must be installed in the computer lab
    - Network maintenance and management

Assessment Guidelines

This section provides the lecturer with guidelines to develop and implement a coherent, integrated assessment system for the subject **Technical Programming** in the National Certificates (Vocational). It must be read with the National Policy Regarding Further Education and Training Programmes: Approval of the Documents, Policy for the National Certificates (Vocational) Qualifications at Levels 2 to 4 on the National Qualifications Framework (NQF). This assessment guideline will be used for National Qualifications Framework Levels 2-4.
This section explains the requirements for the internal and external subject assessment. The lecturer must use this document with the Subject Guidelines: Programming to prepare for and deliver the Programming. Lecturers should use a variety of resources and apply a range of assessment skills in the setting, marking, and recording of assessment tasks.

Assessment in the national certificates (vocational)

Assessment in the National Certificates (Vocational) is underpinned by the objectives of the National Qualifications Framework (NQF). These objectives are to:

- Create an integrated national framework for learning achievements.
- Facilitate access to and progression within education, training and career paths.
- Enhance the quality of education and training.
- Redress unfair discrimination and past imbalances and thereby accelerate employment opportunities.
- Contribute to the holistic development of the student by addressing:
    - social adjustment and responsibility;
    - moral accountability and ethical work orientation;
    - economic participation; and
    - nation-building.

The principles that drive these objectives are:

Integration

To adopt a unified approach to education and training that will strengthen the human resources development capacity of the nation.

Relevance

To be dynamic and responsive to national development needs.

Credibility

To demonstrate national and international value and recognition of qualification and acquired competencies and skills.

Coherence

To work within a consistent framework of principles and certification.

Flexibility

To allow for creativity and resourcefulness when achieving Learning Outcomes, to cater for different learning styles and use a range of assessment methods, instruments, and techniques.

Participation

To enable stakeholders to participate in setting standards and co-ordinating the achievement of the qualification.

Access

To address barriers to learning at each level to facilitate students' progress.

Progression

To ensure that the qualification framework permits individuals to move through the levels of the national qualification via different, appropriate combinations of the components of the delivery system.

Portability

To enable students to transfer credits of qualifications from one learning institution and/or employer to another institution or employer.

Articulation

To allow for vertical and horizontal mobility in the education system when accredited pre-requisites have been successfully completed.

Recognition of Prior Learning

To grant credits for a unit of learning following an assessment or if a student possesses the capabilities specified in the outcomes statement.

Validity of assessments

To ensure assessment covers a broad range of knowledge, skills, values and attitudes (SKVAs) needed to demonstrate applied competency. This is achieved through:

- clearly stating the outcome to be assessed;
- selecting the appropriate or suitable evidence;
- matching the evidence with a compatible or appropriate method of assessment; and
- selecting and constructing an instrument(s) of assessment.

Reliability

To assure assessment practices are consistent so that the same result or judgment is arrived at if the assessment is replicated in the same context. This demands consistency in the interpretation of evidence; therefore, careful monitoring of assessment is vital.

Fairness and transparency

To verify that no assessment process or method(s) hinders or unfairly advantages any student. The following could constitute unfairness in assessment:

- Inequality of opportunities, resources or teaching and learning approaches
- Bias based on ethnicity, race, gender, age, disability or social class
- Lack of clarity regarding Learning Outcome being assessed
- Comparison of students' work with other students, based on learning styles and language

Practicability and cost-effectiveness

To integrate assessment practices within an outcomes-based education and training system and strive for cost and time-effective assessment.

Assessment framework for vocational qualifications

The assessment structure for the National Certificates (Vocational) qualification is as follows:

Internal continuous assessment (ICASS)

Knowledge, skills values, and attitudes (SKVAs) are assessed throughout the year using assessment instruments such as projects, tests, assignments, investigations, role-play and case studies. The internal continuous assessment (ICASS) practical component is undertaken in a real workplace, a workshop or a "Structured Environment". This component is moderated internally and externally quality assured by Umalusi. All internal continuous assessment (ICASS) evidence is kept in a Portfolio of Evidence (PoE) and must be readily available for monitoring, moderation and verification purposes.

External summative assessment (ESASS)

The external summative assessment is either a single or a set of written papers set to the requirements of the Subject Learning Outcomes. The Department of Higher Education and Training administers the theoretical component according to relevant assessment policies.
A compulsory component of external summative assessment (ESASS) is the integrated summative assessment task (ISAT). This assessment task draws on the students' cumulative learning throughout the year. The task requires integrated application of competence and is executed under strict assessment conditions. The task should take place in a simulated or "Structured Environment". The integrated summative assessment task (ISAT) is the most significant test of students' ability to apply their acquired knowledge.
External summative assessments will be conducted annually between October and December, with provision made for supplementary sittings.

Moderation of assessment

Internal moderation

Assessment must be moderated according to the internal moderation policy of the Technical and Vocational (TVET) college. Internal college moderation is a continuous process. The moderator's involvement starts with the planning of assessment methods and instruments and follows with continuous collaboration with and support to the assessors. Internal moderation creates common understanding of Assessment Standards and maintains these across vocational programmes.

External moderation

External moderation is conducted by the Department of Higher Education and Training, Umalusi and, where relevant, an Education and Training Quality Assurance (ETQA) body according to South African Qualifications Authority (SAQA) and Umalusi standards and requirements.

The external moderator:

- monitors and evaluates the standard of all summative assessments;
- maintains standards by exercising appropriate influence and control over assessors;
- ensures proper procedures are followed;
- ensures summative integrated assessments are correctly administered;
- observes a minimum sample of ten (10) to twenty-five (25) percent of summative assessments;
- gives written feedback to the relevant quality assuror; and
- moderates in case of a dispute between an assessor and a student.

Policy on inclusive education requires that assessment procedures for students who experience barriers to learning be customised and supported to enable these students to achieve their maximum potential.

Period of validity of internal continuous assessment (ICASS)

The period of validity of the internal continuous assessment mark is determined by the National Policy on the Conduct, Administration and Management of the Assessment of the National Certificates (Vocational). The internal continuous assessment (ICASS) must be re-submitted with each examination enrolment for which it constitutes a component.

Assessor requirements

Assessors must be subject specialists and should ideally be declared competent against the standards set by the ETDP SETA. If the lecturer conducting the assessments has not been declared a competent assessor, an assessor who has been declared competent may be appointed to oversee the assessment process to ensure the quality and integrity of assessments.

Types of assessment

Assessment benefits the student and the lecturer. It informs students about their progress and helps lecturers make informed decisions at different stages of the learning process. Depending on the intended purpose, different types of assessment can be used.

Baseline assessment

At the beginning of a level or learning experience, baseline assessment establishes the knowledge, skills, values and attitudes (SKVAs) that students bring to the classroom. This knowledge assists lecturers to plan learning programmes and learning activities.

Diagnostic assessment

This assessment diagnoses the nature and causes of learning barriers experienced by specific students. It is followed by guidance, appropriate support and intervention strategies. This type of assessment is useful to make referrals for students requiring specialist help.

Formative assessment

This assessment monitors and supports teaching and learning. It determines student strengths and weaknesses and provides feedback on progress. It determines if a student is ready for summative assessment.

Summative assessment

This type of assessment gives an overall picture of student progress at a given time. It determines whether the student is sufficiently competent to progress to the next level.

Planning assessment

An assessment plan should cover three main processes:

Collecting evidence

The assessment plan indicates which Subject Outcomes and Assessment Standards will be assessed, what assessment method or activity will be used and when this assessment will be conducted.

Recording

Recording refers to the assessment instruments or tools with which the assessment will be captured or recorded. Therefore, appropriate assessment instruments must be developed or adapted.

Reporting

All the evidence is put together in a report to deliver a decision for the subject.

Methods of assessment

Methods of assessment refer to who carries out the assessment and includes lecturer assessment, self- assessment, peer assessment and group assessment.

LECTURER ASSESSMENT	The lecturer assesses students' performance against given criteria in different contexts, such as individual work, group work, etc.
SELF-ASSESSMENT	Students assess their own performance against given criteria in different contexts, such as individual work, group work, etc.
PEER ASSESSMENT	Students assess another student's or group of students' performances against given criteria in different contexts, such as individual work, group work, etc.
GROUP ASSESSMENT	Students assess the individual performance of other students within a group or the overall performance of a group of students against given criteria.

Instruments and tools for collecting evidence

All evidence collected for assessment purposes is kept or recorded in the student's Portfolio of Evidence (PoE).
The following table summarises a variety of methods and instruments for collecting evidence. A method and instrument are chosen to give students ample opportunity to demonstrate the Subject Outcome has been attained. This will only be possible if the chosen methods and instruments are appropriate for the target group and the Specific Outcome being assessed.

	METHODS FOR COLLECTING EVIDENCE		
	Observation-based (Less structured)	**Task-based** (Structured)	**Test-based** (More structured)
Assessment instruments	• Observation • Class questions • Lecturer, student, parent discussions	• Assignments or tasks • Projects • Investigations or research • Case studies • Practical exercises • Demonstrations • Role-play • Interviews	• Examinations • Class tests • Practical examinations • Oral tests • Open-book tests

Assessment tools	• Observation sheets • Lecturer's notes • Comments	• Checklists • Rating scales • Rubrics	• Marks (e.g. %) • Rating scales (1-7)
Evidence	• Focus on individual students • Subjective evidence based on lecturer observations and impressions	**Open middle**: Students produce the same evidence but in different ways. **Open end**: Students use same process to achieve different results.	Students answer the same questions in the same way, within the same time.

Tests

A test could be a practical test (design and development oriented) or a written test. The programme of assessment should reflect a balance between practical and written tests. Tests could include open book tests.
A test for formal assessment should not comprise of a series of small tests but should cover a substantial amount of content and the duration should be a minimum of 60 minutes.
Open book tests require learners to find information and apply knowledge and skills. Learners are tested on understanding and application of learning material and not on rewriting. Open book tests should not include only short questions. They must include questions/tasks that will encourage thinking and decision making.
For written open book tests, learners are required to write longer reflective answers, such as paragraph type responses to a given scenario. Paragraphs providing reasons and supporting evidence/arguments are essential.
For practical open book tests learners are required to apply a combination of a series of procedures and techniques to new situations in order to provide a specific answer or accomplish a specific goal.

Alternative Assessment

Alternative assessment is an alternative to standard tests and exams. It provides a true evaluation of what the learner has learned, going beyond acquired knowledge by looking at their application of this knowledge.

Integrated task/test

An integrated task/test requires learners to be able to apply their knowledge and skills in both theory and practical work that was covered. Testing these types of scenarios e.g. database theory together with database practical, algorithm with implementation and using a trace table to debug a programme.

Case study

Case studies are investigations of real-life situation or simulation thereof. Data is gathered from a variety of sources and by using several different methods. A case study requires an in-depth, and detailed examination of a scenario, as well as the related contextual conditions.
Each test, open book test, alternative assessment task and examination must reflect different cognitive levels.

Tools for assessing student performance

Rating scales are marking systems where a symbol (such as 1 to 7) or a mark (such as 5/10 or 50%) is defined in detail. The detail is as important as the coded score. Traditional marking, assessment and evaluation mostly used rating scales without details such as what was right or wrong, weak or strong, etc.
Task lists and checklists show the student what needs to be done. These consist of short statements describing the expected performance in a particular task. The statements on the checklist can be ticked off when the student has adequately achieved the criterion. Checklists and task lists are useful in peer or group assessment activities.
Rubrics are a hierarchy (graded levels) of criteria with benchmarks that describe the minimum level of acceptable performance or achievement for each criterion. Using rubrics is a different way of assessing and cannot be compared to tests. Each criterion described in the rubric must be assessed separately. Mainly two types of rubrics, namely holistic and analytical, are used.

Selecting and/or designing recording and reporting systems

The selection or design of recording and reporting systems depends on the purpose of recording and reporting student achievement. Why particular information is recorded and how it is recorded determine which instrument will be used.
Computer-based systems, for example spreadsheets, are cost and time effective. The recording system should be user-friendly, and information should be easily accessed and retrieved.

Competence descriptions

All assessment should award marks to evaluate specific assessment tasks. However, marks should be awarded against rubrics and not be simply a total of ticks for right answers. Rubrics should explain the competence level descriptors for the skills, knowledge, values and attitudes (SKVAs) that a student must demonstrate to achieve each level of the rating scale.
When lecturers or assessors prepare an assessment task or question, they must ensure that the task or question addresses an aspect of a Subject Outcome. The relevant Assessment Standard must be used to create the rubric to assess the task or question. The descriptions must clearly indicate the minimum level of attainment for each category on the rating scale.

Strategies for collecting evidence

Several different assessment instruments may be used to collect and record evidence. Examples of instruments that can be (adapted and) used in the classroom include:

Record sheets

The lecturer observes students working in a group. These observations are recorded in a summary table at the end of each project. The lecturer can design a record sheet to observe students' interactive and problem- solving skills, attitudes towards group work and involvement in a group activity.

Checklists

Checklists should have clear categories to ensure that the objectives are effectively met. The categories should describe how the activities are evaluated and against what criteria they are evaluated. Space for comments is essential.

Schedule of assessment

At NQF levels 2, 3 and 4, lecturers will conduct assessments as well as develop a schedule of formal assessments that will be undertaken in the year. All three levels also have an external examination that accounts for 50 percent of the total mark. The marks allocated to assessment tasks completed during the year, kept or recorded in a Portfolio of Evidence (PoE), account for the other 50 percent. The Portfolio of Evidence (PoE) and the external assessment include practical and written components. The practical assessment in Principles of Computer Programming must, where necessary, be subjected to external moderation by Umalusi or an appropriate Education and Training Quality Assurance (ETQA) body, appointed by the Umalusi Council in terms of Section 28(2) of the General and Further Education and Training Quality Assurance Act, 2001 (Act No. 58 of 2001).

Recording and reporting

The subject Robotics, as is the case for all the other Vocational subjects, is assessed according to five levels of competence. The level descriptions are explained in the following table.

RATING CODE	RATING	MARKS %
5	Outstanding	80-100
4	Highly competent	70-79
3	Competent	50-69
2	Not yet competent	40-49
1	Not achieved	0-39

The programme of assessment should be recorded in the Lecturer's Portfolio of Assessment for each subject. The following at least should be included in the Lecturer's Assessment Portfolio:

- A content pages
- The formal schedule of assessment
- The requirements for each assessment task
- The tools used for each assessment task
- Recording instrument(s) for each assessment task
- A mark sheet and report for each assessment task

The college must standardise these documents.
The student's Portfolio of Evidence (PoE) must include at least:

- A content page
- The assessment tasks according to the assessment schedule

Specifications for external assessment in Programming

Background

The National Certificate Vocational (NC (V)) qualification comprises an internal and an external assessment component which both possess a weighting of 50%. The internal assessment component is made up of the internal continuous assessment (ICASS) tasks only, while the external assessment component is made up of the Integrated Summative Assessment Task (ISAT) and an external examination for vocational subjects.

Component		Weighting per component
		Task
		Weighting per task
Internal assessment	50%	ICASS 50%
External assessment	50%	ISAT 15%
		External examination 35%

Practical assessments form part of both the ICASS and ISAT tasks of the NC (V) qualification. While the ISAT accounts for 15% of the final subject mark, it is a common standardised practical task as it is externally set by the Department. The five assessments comprising the ICASS component of the vocational subjects accounting for 50% of the final subject mark are however internally set and therefore vary in standard from one institution to the next. Two of these five ICASS assessments are practical in nature and make up 50% of the ICASS mark which represents 25% of the final subject mark.

ICASS Task		Quantity
		Weighting of ICASS
Tests	2	10% × 2 = 20%
Practical assessments	2	25% × 2 = 50%
Internal Examination	1	30%

In 2017, the two practical assessment tasks forming part of the ICASS were standardised to ensure a uniform standard across institutions and to improve the chances of employment of NC (V) graduates in the workplace. New Subject ISATs were simultaneously introduced in order to facilitate unhindered progression between NC (V) levels in accordance with the NC (V) policy on admission and progression.

The NC (V) policy further requires that assessment tasks must be reviewed on a regular basis to ensure continual relevance, credibility, validity and fairness. The initial practical assessment tasks have now been reviewed to assess new curriculum content introduced in 2019. The two reviewed practical ICASS assessments as well as the reviewed ISAT contained in this document must be implemented in 2019.
The practical assessments for the ICASS and ISAT tasks are to be implemented as a series of three practical tasks per vocational subject to ensure that the practical competencies prescribed per subject are assessed in an authentic practical context which focuses on applied competence.

Task No.	Practical Assessment
1	ICASS PAT 1
2	ICASS PAT 2
3	ISAT

Purpose and value of practical assessments to the workplace

The practical ISAT and ICASS assessments play a central role in expressing the vocational nature of the NC (V) qualification as they require the demonstration of the practical application of theoretical knowledge through the performance of assessment tasks that replicate or simulate a workplace or

real-life process and/or product. Performance in these practical assessments is therefore used by industry as a measure to determine readiness of NC (V) graduates to enter the workplace.

Integrated summative assessment task (ISAT)

A compulsory component of the external assessment (ESASS) is the integrated summative assessment task (ISAT). The integrated summative assessment task (ISAT) draws on the students' cumulative learning achieved throughout the year. The task requires integrated application of competence and is executed and recorded in compliance with assessment conditions.

Principles for the conduct of practical assessments

The following principles are applicable to the conduct of practical assessments:
Each student must be provided with a copy of a subject assessment schedule which includes all three practical assessments (see ICASS Guidelines).

The criteria to be used to assess student performance in a practical assessment must be made available to the students and be explained prior to the conduct of each practical assessment.

All practical assessments, i.e. performance based and/or creation of product/artefact, must be undertaken in controlled conditions under the direct supervision of the subject lecturer.

An internal moderator must moderate at least 10% or a minimum of 5 performance-based assessments during the actual performance.

Practical assessments must take place in suitable environments relevant to the task specifications, e.g. workshop, computer laboratory, simulator or actual workplace.

National Examination

A national examination is conducted annually in October or November by means of a paper(s) set and moderated externally. The following distribution of cognitive application is suggested:

Cognitive and difficulty levels of formal assessments

Formal assessments must cater for a range of cognitive levels and abilities of learners as shown in the table below:

Cognitive Level	Taxonomy	Description
C1	Knowledge, Remembering	Recall of factual/process knowledge **in isolation**, i.e. one step/ set of basic steps/instruction/process at a time, e.g. definitions in the theory paper and known procedures/algorithms in the practical paper. It also presents the knowledge of the various theoretical components and subject content relating to the composition and design of applications. These include: - knowledge about the various components and constituents of an application - design related aspects and considerations

		- knowledge of the applicability and application of formulas
C2	Understanding, Applying	Demonstrates **understanding** of: - steps/algorithms/processes/ isolatable bits, such as translating from one form of representation to another, e.g. converting a flow chart representation of a program/program segment to a functional program. - Various components and their applications, uses, and common properties It also requires using **known routines/algorithms/processes/design constructs and components** in a familiar context in order to complete a task, where **all of the information required is immediately available to the student**.
C3	Analysing, Evaluating, Creating	Requires **reasoning/investigation/developing a plan** or sequence of steps/algorithm or an electronics related artefact in the form of a component; has some complexity where candidates need to see how parts relate to a whole; organising/ putting together component parts/elements to form a coherent functional whole/achieve an overall objective and completing a task could have more than one possible approach. It could also require weighing possibilities, deciding on the most appropriate solution, as well as testing to locate errors/ troubleshooting, pattern recognition and generalisation. These questions will comprise actions/strategies/procedures where candidates are required to create their own solutions to challenges, they may encounter. These questions could include analysing questions or data, and decision-making.

Levels of difficulty (D) are categorised as follows:

D1: Easy for the average student (in relation to the subject Level 2 – 4) to answer

D2: Moderately challenging for the student (in relation to the subject Level 2 – 4) to answer

D3: Difficult for the average student (in relation to the subject Level 2 – 4) to answer

D4: Very difficult for the average student (in relation to the subject Level 2 – 4) to answer. The skills and knowledge required to answer questions at this level should be included to distinguish amongst high achievers.

Questions in the formal assessment tasks will assess performance at different cognitive levels, critical thinking skills, problem-solving techniques and difficulty, as outlined below.

In judging the level of difficulty of each question, both the demands that each question makes on the cognitive ability of an average Programming student and the intrinsic level of difficulty of the question or task are considered. In making this judgement, the difficulty or ease of a particular question is identified. A four-category framework for thinking about question or item difficulty adapted from Leong (2006) has been used in this identification process. This framework comprises the following four general categories of difficulty:

- **Content difficulty:** This indexes the difficulty of the subject matter, topic or conceptual knowledge; some content is inherently more difficult than other content.
- **Stimulus difficulty:** This relates to the linguistic features of the question and the challenge that candidates face in reading, interpreting and understanding the question.
- **Task difficulty:** This refers to the difficulty that candidates face when trying to formulate or produce an answer.
- **Expected response difficulty:** This refers to difficulties because of the mark scheme or marking guidelines, in other words how marks are to be allocated.

Weighting of cognitive levels and difficulty levels

Papers 1 and 2 will include questions across three cognitive levels. The distribution of cognitive levels in the practical and theory papers is given in the table below.

Cognitive Level	Description	Paper 1 (Theory)	Paper 2 (Design related)
1	Knowledge and remembering	40%	40%
2	Understanding and applying	40%	40%
3	Analysing, evaluating and creating	20%	30%

The estimated percentages for each level of difficulty (D) within each cognitive level (C) are shown in the table below.

	D1	D2	D3	D4	TOTAL
C1	±10%	±10%	±10%	-	**±30%**
C2	±15%	±15%	±8%	±2%	**±40%**
C3	±15%	±7%	±5%	±3%	**±30%**
TOTAL	**±40%**	**±32%**	**±23%**	**±5%**	**100%**

Students are required to investigate and analyse problems in a variety of contexts (such as scientific, technological, environmental and everyday-life contexts) in order to solve the described problems effectively, either via design and development of a solution artefact in Paper 2 or describe proposed solutions in Paper 1/Paper 2.

Paper types

Paper	Type
1	 **Theory**
2	 **Design / Practical / Programming related)**

Examinations

Paper suggested marks and time

Level	Paper 1 (Theory)	Paper 2 (Design / Practical / Programming related) PRACTICAL EXAM ON A COMPUTER
2	100 Marks 2 Hours	100 Marks 3 ½ Hours
3	120 Marks 3 Hours	100 Marks 3 ½ Hours
4	120 Marks 3 Hours	120 Marks 4 Hours

Suggested topic distribution

Level	Paper 2 - Practical Practical exam on computer (Design / Solution development and coding)		Paper 1 (Theory / Algorithm development, short paper-based programming questions)	
2	**TOPIC**	**WEIGHT**	**TOPIC**	**WEIGHT**
	1	2%	1	20%
	2	18%	2	25%
	3	20%	3	10%
	4	10%	4	10%
	5 to 10 (Solution Development)	50%	5 to 10 (Solution Development)	35%

Level	Paper 2 - Practical Practical exam on computer (Design / Solution development and coding)		Paper 1 (Theory / Algorithm development, short paper-based programming questions)	
3	TOPIC	WEIGHT	TOPIC	WEIGHT
	1	2%	1	20%
	2	18%	2	20%
	3	20%	3	10%
	4	10%	4	10%
	5 to 12 (Solution Development)	50%	5 to 12 (Solution Development)	40%

Level	Paper 2 - Practical Practical exam on computer (Design / Solution development and coding)		Paper 1 (Theory / Algorithm development, short paper-based programming questions)	
4	TOPIC	WEIGHT	TOPIC	WEIGHT
	1	2%	1	20%
	2	13%	2	20%
	3	5%	3	10%
	4	5%	4	10%
	5 to 11 (Solution Development)	75%	5 to 11 (Solution Development)	40%

Composition of the different papers

Paper 1 - (Theory)

The paper will cover all theory aspects of all content, concepts and skills of topics, including of software development, e.g. algorithmic development, data structures, program design and general programming concepts as well as generic problem-solving questions.
Typical questions include short type questions pertaining to programming language content and syntax, debugging and short code snippets.

The paper does not have an overarching scenario. Each question may have its own scenario.

Paper 2 - (Design / Practical / Programming related related)

This will be a practically oriented paper covering questions on software and program development. Students will receive 30 minutes reading and planning time before commencement of the practical examination. Students will also receive a script to use for planning and notes during this period and the duration of the examination. A particular question may also require the students to complete some formal planning e.g. flow chart / pseudo-code / algorithm for submission
To successfully complete this paper, each student must have access to his or her own computer in the exam room. Provision needs to be made for sufficient computers to enable the examination to be completed in 2 sittings.
This paper assesses the practical skills as well as the knowledge and understanding underlying the skills pertaining to software development, i.e. the high-level programming language studied.
The paper does not have an overarching scenario. Each question may have its own scenario.
The paper will comprise questions covering the following broad topics:

- Basic programming skills
- General problem solving – different levels of higher order skills

Software design tools may be provided as part of the problem statement (IPO tables, basic flow charts and class diagrams).
The learner will not be required to enter large amounts of data. The required data could be retrieved from the data disk or imported from documents such as a text file, or a database table. All GUIs will be provided.
Marks for questions must be allocated towards basic skills, concepts, constructs and problem-solving techniques, e.g. application of an iteration structure as part of the solution (correct structure) as well as for the correct use of the structure. The allocation of marks should consider the time spent on solving, coding and debugging a solution.

Final exam mark

(Mark for Paper 1 + Paper 2) / (Combined paper total) x 100

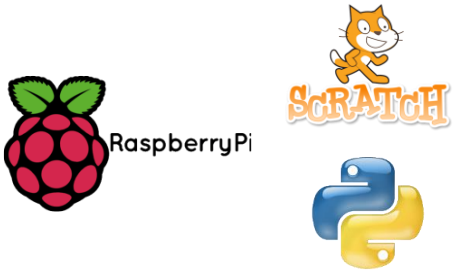
	L2 Basic principles of Computer programming and Computer literacy	L3 Introduction to Technical Programming	L4 Technical Programming
Topic 1 **Computer hardware and software**  The follwing icon  indicates integration with other SO's and LO's	1.1 **Hardware** Content: • Computer components • Hardware classifications *Learning Outcomes:* *Students should be able to:* 1.1.1. Describe the term: Hardware 1.1.2. Describe how hardware communicate: *(Range: electrical signals (analogue vs binary) signals)* 1.1.3. Classify different hardware components in relation to the evolution of technology. *(Range: Processing hardware, Storage hardware, input output devices, and communication devices)* 1.1.4. List different hardware components with relation to their use: Input, Processing, Storage, Output and Transmission 1.2 **Hardware components of a typical system** Content: • Input, Processing, Storage and Output hardware • Systems unit and components *Learning Outcomes:* *Students should be able to:* 1.2.1 Define the term input hardware and provide some examples of each 1.2.2 Describe the purpose of input hardware 1.2.3 Describe the purpose of output hardware 1.2.4 Describe the purpose of processing hardware 1.2.5 Describe the purpose of storage hardware 1.2.6 Define the term processing hardware and provide some examples of each	1.1 **Different types of hardware of a common system** Content: • System Unit • Motherboard • CPU • Memory • Input and Output hardware *Learning Outcomes:* *Students should be able to:* 1.1.1 Identify the components of the system unit 1.1.2 Explain the term CPU and its purpose 1.1.3 Describe the term CPU and explain the impact of using various different types of CPU's 1.1.4 Describe different types of CPU's with regard to use, and power 1.1.5 Define the term computer memory 1.1.6 Discuss the primary purpose of memory 1.1.7 Differentiate between different types of memory and their purpose (Range: RAM, ROM, CMOS, Cache memory, Flash Memory) 1.1.8 Explain how data is stored on memory 1.1.9 Define the purpose of the motherboard and its components 1.1.10 Describe different types of input hardware with regard to use and classification e.g. direct and indirect entry 1.1.11 Discuss how data is transferred between memory i.e. primary and secondary and the CPU 1.1.12 Describe the Flow/transfer of data between components. (Range: USB – PnP, U3, Point-to-point connections) 1.1.13 Describe the factors to consider when choosing an input device. (Range: Ergonomic considerations, Wireless vs cables) 1.1.14 Describe different types of output hardware	1.1 **Computer memory** Content: • Computer Memory • Electronic data storage *Learning Outcomes:* *Candidates should be able to:* 1.1.1 Define the term computer memory 1.1.2 Describe how data is stored electronically 1.1.3 Differentiate between various types of memory with regard to use, purpose and size. (Range: RAM, ROM, Cache, Flash Memory, CMOS) 1.1.4 Describe the Purpose and role of cache memory and caching 1.2 **Application software** Content: Application Software Different types of applications and their use *Learning Outcomes:* *Candidates should be able to:* 1.2.1 Define the term application software 1.2.2 Discuss the purpose of application software 1.2.3 Define the concept of an App in relation to a mobile or standalone device 1.2.4 Discuss and differentiate between different types of application software and their purpose. (Range: Business Software – Office Suites / Productivity suites, database software, note taking software, Accounting and Payroll software, Graphical suites.) Software for home or personal use: - Video editors, games, finance software, educational software, personal information management software)

	1.2.7 Define the term output hardware and provide some examples of each 1.2.8 Describe the typical components of the systems unit 1.2.9 Discuss the concept: Modular design of a computer-based system 1.2.10 Define the term single board computer 1.2.11 List examples of single board computers 1.2.12 Discuss the purpose and uses of single board computers 1.2.13 Present a typical breakdown of the components of a single board computer 1.3 **The Windows command prompt** Content: • Working with the Windows command prompt Learning Outcomes: *Students should be able to:* 1.3.1 Define the term shell 1.3.2 Launch a new command prompt in Windows 1.3.3 Use the help command to list some common commands 1.3.4 Expand a Windows file path and explain each element 1.3.5 List the contents of the current folder using the dir command 1.3.6 Change directly location using the cd command 1.3.7 Create a new folder using the mkdir command 1.3.8 Remove a folder using the rmdir command 1.3.9 Remove a file using the del command 1.3.10 Rename a file using the rename command 1.3.11 Copy a file using the copy command 1.3.12 Clear the command prompts screen using the cls command 1.3.13 Run an executable file from the command line 1.3.14 List the system info using the systeminfo command 1.3.15 Copy the path of any file by dropping the file in the command prompt	1.1.15 Describe the purpose and use of devices such as docking stations for mobile and laptop computers. 1.2 **Purpose of software** Content: • Software as a component of a computerised system • Categories of software Learning Outcomes: *Students should be able to:* 1.2.1 Describe the term software 1.2.2 Describe the purpose and function of software 1.2.3 Describe the basic concepts of software. (Range: Software as programs. Identify software components. Concept of a graphical user interface (GUI).) 1.2.4 Contrast: System software vs application software 1.2.5 Differentiate between: Shareware, Freeware, Open Source Software and Proprietary software, Firmware 1.2.6 Discuss the process of how software is obtained and installed. 1.2.7 Differentiate between online software and installed software. 1.2.8 Discuss the following terms in relation to software. (Range: Compatibility issues, Versions, patches and service packs, Updating software) 1.3 **The Linux shell** Content: • Working with the Linux Bash terminal Learning Outcomes: Students should be able to: 1.3.1 Launch a new Linux terminal on the Raspberry Pi 1.3.2 Use the man command to get help 1.3.3 Expand a Linux file path and explain each element 1.3.4 List the contents of the current folder using the ls command 1.3.5 Change directly location using the cd command	1.2.5 Describe the concept of web and online applications 1.2.6 List examples of web applications and describe their use. 1.2.7 Discuss the concept of a programming language as an example of an application software 1.2.8 List and differentiate between different types of programming languages. (Range: Compilers, Interpreters, Scripting Languages) 1.3 **System software** Content: System Software Operating system *Learning Outcomes:* *Students should be able to* 1.3.13 Define the term system software 1.3.14 Discuss the purpose of system software 1.3.15 Define the term operating system 1.3.16 Discuss the purpose of an operating system 1.3.17 List and describe different types of operating systems and their benefits 1.3.18 Differentiate between free operating systems and proprietary OS's 1.3.19 Describe the main functions of an operating system. (Range: General role: Suite/group of related programs which manage hardware and software. Specific role: Provides user Interface, I/O management, device management, task management, performance management.) 1.3.20 Discuss the role of the operating system in terms of file, disk, memory, and storage and process management.

		1.3.6 Create a new folder using the mkdir command 1.3.7 Remove a folder using the rmdir command 1.3.8 Remove a file using the rm command 1.3.9 Rename a file using the mv command 1.3.10 Copy a file using the cp command 1.3.11 Clear the command prompts screen using the cls command 1.3.12 Run an executable file from the command line	

Topic 2 **Problem solving in computer programming** Note 2.1 and 2.2 do repeat but problems to be solved will increase in complexity based on the level expected.	2.1 **Problem solving process and concepts** Content: • Problem solving • Application of problem-solving constructs • Developing solutions *Learning Outcomes:* *Students should be able to:* 2.1.1 Define the term problem solving 2.1.2 Define the term computational thinking 2.1.3 Describe the phases of the PLDC (Program Development Life Cycle) 2.1.4 Describe the purpose of problem solving leading to solutions 2.1.5 Explain and apply various problem-solving steps. Polya, G., 1957) (Range:Understand the problem (task/problem description or scenario/user stories) State in own words Clarity on what needs to be done What is known or given? What is missing or needed? Devise a plan/algorithm (storyboard – visual or textual) Look for patterns Look at related problems, known solutions Examine simpler or special cases Make a table, create diagram, use guess and check, work backwards, identify sub-goal Carry out the plan/implement the algorithm (write the code) Look back/test (see if it works) Check results against original problem. Does it make sense? Is there another solution?) 2.1.6 Use appropriate tools and techniques to present a solution. Range: User stories (written by the client and provide the requirements) Noun-verb analysis of user stories List of nouns provides identification of objects and state List of verbs provides identification of behaviour Acceptance tests (does the program meet the requirements?)	2.1 **Problem solving process and concepts** Content: • Problem solving • Application of problem-solving constructs • Developing solutions *Learning Outcomes:* *Students should be able to:* 2.1.1 Define the term problem solving 2.1.2 Define the term computational thinking 2.1.3 Describe the phases of the PLDC (Program Development Life Cycle) 2.1.4 Describe the purpose of problem solving leading to solutions 2.1.5 Explain and apply various problem-solving steps. Polya, G., 1957) (Range:Understand the problem (task/problem description or scenario/user stories) State in own words Clarity on what needs to be done What is known or given? What is missing or needed? Devise a plan/algorithm (storyboard – visual or textual) Look for patterns Look at related problems, known solutions Examine simpler or special cases Make a table, create diagram, use guess and check, work backwards, identify sub-goal Carry out the plan/implement the algorithm (write the code) Look back/test (see if it works) Check results against original problem. Does it make sense? Is there another solution?) 2.1.6 Use appropriate tools and techniques to present a solution. Range: User stories (written by the client and provide the requirements) Noun-verb analysis of user stories List of nouns provides identification of objects and state List of verbs provides identification of behaviour Acceptance tests (does the program meet the requirements?)	2.1 **Problem solving process and concepts** Content: • Problem solving • Application of problem-solving constructs • Developing solutions *Learning Outcomes:* *Students should be able to:* 2.1.1 Define the term problem solving 2.1.2 Define the term computational thinking 2.1.3 Describe the phases of the PLDC (Program Development Life Cycle) 2.1.4 Describe the purpose of problem solving leading to solutions 2.1.5 Explain and apply various problem-solving steps. Polya, G., 1957) (Range:Understand the problem (task/problem description or scenario/user stories) State in own words Clarity on what needs to be done What is known or given? What is missing or needed? Devise a plan/algorithm (storyboard – visual or textual) Look for patterns Look at related problems, known solutions Examine simpler or special cases Make a table, create diagram, use guess and check, work backwards, identify sub-goal Carry out the plan/implement the algorithm (write the code) Look back/test (see if it works) Check results against original problem. Does it make sense? Is there another solution?) 2.1.6 Use appropriate tools and techniques to present a solution. Range: User stories (written by the client and provide the requirements) Noun-verb analysis of user stories List of nouns provides identification of objects and state List of verbs provides identification of behaviour Acceptance tests (does the program meet the requirements?)

	2.2 **Construct an algorithm and present a solution to a given problem** Content: • Problem solving • Algorithm design • IPO Chart • Flowchart *Learning Outcomes:* *Students should be able to:* 2.2.1 Define the term algorithm and its purpose in the problem-solving process. (Range: Basic concepts of an algorithm. What is an algorithm? Develop a clear understanding of the problem presented.) 2.2.3 List examples of algorithms in life, e.g. instructions to draw a kite or fold a paper jet, recipe to bake a cake 2.2.4 Construct and devise an algorithm/basic instruction to complete similar tasks. 2.2.5 Describe the purpose of input, processing and output as part of the solution creation process 2.2.6 Define the term **IPO chart** and explain its purpose in solution development 2.2.7 Outline the various parts of an IPO chart 2.2.8 Define the term **flow chart** and explain it purpose in solution development 2.2.9 Outline the various symbols used as part of a **flow chart** 2.2.10 Explore and create algorithms in the form of an **IPO chart** and a flowchart which include various constructs, including: - Sequence - Selection - Iteration	2.2 **Construct an algorithm and present a solution to a given problem** Content: • Problem solving • Algorithm design • Flowcharts *Learning Outcomes:* *Students should be able to:* 2.2.1 Define the term algorithm and its purpose in the problem-solving process. (Range: Basic concepts of an algorithm. What is an algorithm? Develop a clear understanding of the problem presented.) 2.2.2 Implement and understand the basic algorithmic constructs used to create a **flowchart.** Range: Input, Output, Processing and Calculations, Selection Iteration 2.2.3 Create a flowchart to present a particular algorithm and its associated tasks 2.2.4 Interpret a basic flow chart and describe its intended operation / function	2.2 **Construct an algorithm and present a solution to a given problem** Content: • Problem solving • PLDC • IPO Chart *Learning Outcomes:* *Students should be able to:* 2.2.1 Define the term algorithm and its purpose in the problem-solving process. (Range: Basic concepts of an algorithm. What is an algorithm? Develop a clear understanding of the problem presented.) 2.2.2 Extrapolate the position of algorithm development as part of the PLDC (Program Development Life Cycle) 2.2.3 Implement and understand the basic algorithmic constructs used to create pseudo code algorithm. Range: Input, Output, Processing and calculations, selection, and iteration 2.2.4 Explore and create algorithms in the form of an IPO chart or flowchart which include various constructs, such as: 2.2.4.1 Determine smallest, largest value of more than two values 2.2.4.2 Swapping values 2.2.4.3 Determining aggregates e.g. sum and average 2.2.4.4 Basic calculations such as calculating area, volume, VAT 2.2.4.5 Determine whether a number is even 2.2.4.6 Determine whether a number is a factor of another number 2.2.4.7 Determine whether a number is prime or not 2.2.4.8 Determine and print whether a number is part of the Fibonacci series 2.2.4.9 Print the first n^{th} set of cube numbers 2.2.4.10 Convert a decimal number into binary and vice versa 2.2.5 Explore and create algorithms in the form of an IPO chart or flowchart which include various constructs, such as:

			2.2.5.1 Populating and array with user defined values 2.2.5.2 Find the largest and smallest value in an array 2.2.5.3 Determine the sum and average of all the elements in an array 2.2.5.4 Reverse the elements in an array 2.2.5.5 Swop two elements in an array 2.2.5.6 Performing a simple sort 2.2.5.7 Split and array and add the first part to the end 2.2.6 Compare algorithms considering, e.g. order, precision and efficiency
Topic 3 **Concepts of programming for single board microprocessor or microcontrollers**			
The following topic is a cross curricular topic and should be integrated with Topic 5 of the Robotics Curriculum where programming is applied enable the students to create projects that	3.1 **Visual Programming and solution development** Content: • Writing program code using a graphical (visual) programming language • Debugging • Compilation • Testing • Running an application *Learning Outcomes:* *Students should be able to:* 3.1.1 Define the term block based / visual programming 3.1.2 List examples of different block based / visual programming languages and their use 3.1.3 Explore and use the visual programming language development environment. (Range: Open an existing	**Repeat from Level 2 (This relates to the GPIO programming and standard programming content covered in L2 to show the relationship and overlap in concepts)** **3.1 Introduction to IO on single board computing** **Content:** • GPIO • Physical computing *Learning Outcomes:* *Students should be able to:* 3.1.1 3.1.2 Expand the term GPIO	3.1 **The ARM architecture** Content: • The ARM Architecture *Learning Outcomes:* *Students should be able to:* 3.1.1 Define the term ARM 3.1.2 Explain the difference between ARM and X86 (Intel/AMD) architecture 3.1.3 Explain the benefits of the ARM architecture 3.1.4 List devices where the ARM architecture is used 3.1.5 Explain how ARM processors are designed and manufactured 3.1.6 Explain how software needs to be adapted to run on the ARM architecture

involve physical computing. 	program I.e. code, Save an existing program. Modify an existing program. Debug an existing program. Create a program, Run a program.) 3.1.4 Construct/write) using the visual tool, debug and run simple programs incorporating: Declaration of variables of different types, use and assignment of values to variables, incorporating program constructs with sequence structures. 3.1.5 Demonstrate a knowledge of various concepts encapsulated in a block-based language such as: 3.1.5.1 Differentiating between a sprite and an image 3.1.5.2 Using existing images and sprites 3.1.5.3 Creating a user defined sprite 3.1.5.4 Different categories of block-based coding blocks (e.g. events, motion, control, variables…. Etc.) 3.1.5.5 Changing appearance and costumes 3.1.5.6 Adding buttons 3.1.5.7 Assigning events and triggers 3.1.5.8 Creating and using variables 3.1.5.9 Operations on data 3.1.5.10 Comparisons 3.1.5.11 Broadcasting 3.1.5.12 Repeating actions 3.1.5.13 Using conditional statements 3.1.5.14 Manipulating strings 3.1.5.15 Basic list operations 3.1.5.16 incorporating sounds and images 3.1.5.17 incorporating multiple sprites 3.1.6 Expose and apply various programming concepts as part of the coded solution such as: 3.1.6.1 the use of variables (defining and naming) 3.1.6.2 variable naming conventions 3.1.6.3 assigning values to variables (assignment operator) 3.1.6.4 data type - exposure: Integers, strings, floats, Boolean 3.1.6.5 input and output of information and massages and values 3.1.6.6 using single or a combination of different operators (plus, minus, multiply, divide) and order of precedence	3.1.3 Compare and contrast some of the major advantages of python compared to other programming languages 3.1.4 Explain and identify where the GPIO pins are located 3.1.5 Differentiate between a compiler and an interpreter 3.1.6 Explain the purpose of the GPIO pins 3.1.7 Discuss the major characteristics of the python programming language as an interpreted one 3.1.8 Define the term physical computing 3.1.9 Differentiate between a shell and an IDE 3.1.10 Read and interpret a Pi GPIO Pin guide 3.1.11 Define the term physical computing 3.1.12 Explain what the terms 3V3, 5V, GND GP2 means on the GPIO board 3.1.13 Discuss how Arduino is used to enable physical computing 3.2 **Visual Programming and solution development** Content: • Writing program code using a graphical (visual) programming language • Debugging • Compilation • Testing • Running an application *Learning Outcomes:* *Students should be able to:* 3.2.1 Construct (code/write) using the visual tool, debug and run simple programs incorporating: Declaration of variables of different types, use and assignment of values to variables, incorporating program constructs with sequence, selection and iteration structures. Expose and apply various programming concepts as part of the coded solution such as: 3.2.1.1 retrieving remainders: modulus 3.2.1.2 differentiate between real value division and integer division 3.2.1.3 comparison operators and performing logical comparisons 3.2.1.4 incorporate and write code constructs to perform basic calculations such as area, volume, VAT and simple formulae,	3.1.7 Elaborate on why the RaspberryPi creators chose to use the ARM architecture 3.2 **Linux remote administration** Content: • Remote access • VNC protocol • SSH protocol *Learning Outcomes:* *Students should be able to:* 3.2.1 Define and explain remote access 3.2.2 Explain different scenarios where remote access to a Raspberry Pi may be required 3.2.3 Define the term SSH and VNC 3.2.4 Explain the purpose of SSH and VNC 3.2.5 Explain how SSH and VNC differs from each other 3.2.6 List the advantages and disadvantages of SSH and VNC respectively 3.3 **Linux remote file editing** Content: • Nano *Learning Outcomes:* *Students should be able to:* 3.3.1 Explain why a terminal editor is useful 3.3.2 Execute the Linux command to install the Nano editor 3.3.3 Create a new file using the touch command 3.3.4 Copy a file to make a backup before editing using the cp command 3.3.5 Open file with the Nano editor 3.3.6 Change file content using Nano editor 3.3.7 Search for a word / phrase (exact match) inside the Nano editor 3.3.8 Quit Nano without saving file changes 3.3.9 Save changes and quit the nano editor

	3.1.6.7 comparison operators and performing logical comparisons 3.1.6.8 incorporate and write code constructs to perform various calculations such as area, volume, VAT and simple formulae, typical calculations done in other subjects 3.1.6.9 include conditional constructs [if and if-then-else] (up to a maximum of three nested levels) 3.1.6.10 include iteration constructs as part of a solution (up to a maximum of two nested levels) 3.1.6.11 incorporating a single list as part of the solution 3.1.6.12 include basic string operations as part of the solution 3.1.7 Write code which applies programming language tools and constructs to draw various shapes (turtle type commands) on an output screen/window. Reinforce concepts such as: 3.1.7.1 Sequence 3.1.7.2 Selection 3.1.7.3 Iteration 3.1.8 Devise a specific algorithm where applicable to solve a problem utilising user-defined code constructs. 3.2. **Install Python on single board microprocessor** **Content:** • Install Python on RaspberryPi • Install IDLE on RaspberryPi • Basic Python program Learning Outcomes: *Students should be able to:* 3.2.1 Execute the commands to install Python 3 on the RaspberryPi 3.2.2 Execute the commands to install the Mu Python editor on the RaspberryPi	typical calculations done in other subjects 3.2.1.5 include conditional constructs [if and if-then-else] (up to a maximum of two nested levels) 3.2.1.6 include iteration (looping) structures [fixed counter loop] 3.2.1.7 incorporate a combination of iteration and condition structures as part of the solution (i.e. program code) 3.2.2 Write code which applies programming language tools and constructs to draw various shapes (turtle type commands) on an output screen/window. Reinforce concepts such as: 3.2.2.1 Sequence 3.2.2.2 Selection 3.2.2.3 Iteration 3.2.2.4 Creation of objects and shapes 3.2.3 Design a coding solution to a problem incorporating a combination of different programming constructs which include: 3.2.3.1 Sequence 3.2.3.2 Selection 3.2.3.3 Iteration 3.2.4 Design and develop solutions for specific problems that include computational thinking and applying software engineering principles. 3.2.5 Explore lists/arrays (storing and accessing a list of numbers and strings) and containers. (Range: Manipulating lists/arrays such as adding, deleting, replacing, inserting items.)	

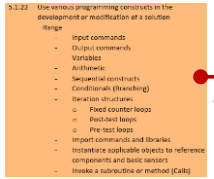
	3.2.3 Start the Mu file editor and create a new Python file. 3.2.4 Add a simple "Hello world" print statement to an open Mu file 3.2.5 Use the Mu editor to create a simple application 3.2.6 Run/Execute the source code 3.2.7 Save the file 3.2.8 Open an existing file. 3.2.9 Open and run/execute simple sample files **3.3. Basic Python applications and the Turtle library** Content: • Basic Turtle setup • Drawing in Turtle • Changing colours Learning Outcomes: *Students should be able to:* 3.3.1 Differentiate between a compiler and an interpreter 3.3.2 Differentiate between a shell and an IDE 3.3.3 Discus the major characteristics of Python as an interpreted programming language 3.3.4 Setup simple turtle Python program 3.3.5 Explain the purpose of the from import statement in Python 3.3.6 Draw a line using the forward function 3.3.7 Turn using the right and left functions 3.3.8 Combine the draw, left and right functions to create simple drawings (rectangle, triangle, cross, alphabet letters etc.) 3.3.9 Explain how a colour is made up of a mix of Red Green and Blue 3.3.10 Change the line colour using the colour function 3.3.11 Combine drawing with changing the colour		
	The aim of topic 3 is to introduce students to programming using a block-based language such as Scratch. In Level 2 of Robotics the use of a Raspberry Pi is prescribed for being used as both a computing device (i.e., a computer) and a device to allow for the development of projects that interfaces with hardware components. The subject outcomes of Topic 2 should therefore be integrated with that of topic 3.		

Topic 4 **Programming tools and utilities**	**4.1. Source control** Content: • Version control software *Learning Outcomes:* *Students should be able to:* 4.1.1 Define the term revision control in general. 4.1.2 Explain the consent of a revision number 4.1.3 Name and explain the two main components of a software revision (timestamp and author) 4.1.4 Explain why there is a need for a logical way to organise and control revisions in software development 4.1.5 Define the term Version Control System (VCS) 4.1.6 Elaborate how a VCS is different than generic revision control 4.1.7 Differentiate between a Branch and a Trunc as used in VCS's 4.1.8 List advantages to splitting development of software into different branches 4.1.9 List the disadvantages to keeping multiple application version copies in separate folders when developing software 4.1.10 Explain the concept of a centralised VCS 4.1.11 Explain the concept of a distributed VCS	**4.1. Source control** Content: • Revisit the concept of version control software and why it is important  **Link with 4.6** **4.2. Building and running C/C++ applications** Content: • The C and C++ compiler • Compiler artefacts • Compiler design *Learning* Outcomes*:* *Students should be able to:* 4.2.1 Define *the* term compiler 4.2.2 Define the term source code 4.2.3 Explain what a decompiler is used for 4.2.4 Define the term interpreter 4.2.5 Explain the difference between a compiler and an interpreter 4.2.6 Explain what a binary is and when it is produced 4.2.7 Explain the difference been C and C++ compiler 4.2.8 List and Explain the basic three stage compiler design (Front Middle Back end) **4.3. C/C++ on the Desktop** Content: • C/C++ compiler environment on desktop PC • IDE on desktop PC *Learning Outcomes:* *Students should be able to:* 4.2.1 Install and configure C/C++ compiler on desktop PC 4.2.2 Define the term IDE 4.2.3 Explain what an IDE is used for 4.2.4 Install and configure IDE on desktop PC	4.1 **Practical Git** Content: • Using Git for version control *Learning Outcomes:* *Students should be able to:* 4.1.1 Verify the Git version installed using the Git version command 4.1.2 Initialise a new Git repository or add existing files to a new Git repository 4.1.3 Add single or multiple files to the repository using the staging command 4.1.4 Commit changes to the repository using the commit command, including adding a commit message 4.1.5 Check the status of a Git repository using the Git status command 4.1.6 Interpret the results of running the Git status command 4.1.7 Run and interpret the results from the Git log command 4.1.8 Create a new branch using the Git branch command 4.1.9 List all branches using the Git branch command 4.1.10 Switch between branches using the Git checkout command 4.2 **Getting started with Python** • Install and configure a Python IDE on desktop PC • Basic project maintenance 4.2.1 Explain the process of installing Python and a high level IDE 4.2.2 Create a new Python project using the IDE 4.2.3 Discuss the difference between a Python project and a Python file. 4.2.4 Use the IDE to create a simple application 4.2.5 Run the source code 4.2.6 Save the project 4.2.7 Open an existing project. 

		4.4. C/C++ on the Raspberry Pi Content: • The GCC C/C++ compiler • Geany on Raspberry Pi *Learning Outcomes:* *Students should be able to:* 4.3.1 Explain what the acronym GCC stands for 4.3.2 Explain what the GCC collection contains 4.3.3 List compilers included in the GCC 4.3.4 Explain the difference between GCC and MinGW 4.3.5 Install and configure GCC C/C++ compiler on Raspberry Pi 4.3.6 Install and configure Geany on Raspberry Pi **4.4. Debugging C/C++ applications** Content: • Debugging • GDB *Learning Outcomes:* *Students should be able to:* 4.4.1 Define the term debugging 4.4.2 Explain why an application needs to be debugged 4.4.3 List common debugging techniques (Interactive, Print, Remote) 4.4.4 Define the term breakpoint as it relates to debugging 4.4.5 Define the term stepping as it relates to debugging 4.4.6 Explain what the GDB tool is used for 4.4.7 Install and configure MinGW on desktop PC 4.4.8 Locate the GDB application in the MINGW installation 4.4.9 Add the MINGW bin directory to the operating system path environment variable 4.4.10 Compile source code wit the -g flag in IDE **4.5. Debugging C/C++ with GDBgui** Content: • GDBgui • C/C++ debugging	PyCharm meets the requirements as a suitable high-level IDE for implementation to cover Topics 4 to 12  PyCharm is an extremely popular Python IDE. An Integrated Development Environment or IDE features a code editor and a compiler for writing and compiling programs in one or many programming languages. Furthermore, an IDE comes with a galore of features that facilitate comprehensive software development. (https://hackr.io/blog/what-is-pycharm)

		Learning Outcomes: *Students should be able to* 4.5.1 Explain the concept of a debugger frontend 4.5.2 Install and configure gdbgiu on Desktop PC 4.5.3 Start gdbgui application 4.5.4 Load a compiled C/C++ binary (.exe) 4.5.5 Run the loaded binary 4.5.6 Step through the running binary line by line 4.5.7 Investigate (watch) the values of local variables **4.6. The Git Version Control System** Content: • Git concepts 4.6.1 Explain what Git is and what Git is used for 4.6.2 Discuss the three main goals of the Git VCS - speed - data integrity - distributed 4.6.3 Explain the relationship between local Git directory and directory located on server 4.6.4 Discuss the software license used for Git 4.6.5 Name and explain the advantages and disadvantages to the software license used by Git 4.6.6 Explain why Git was created 4.6.7 Explain the relationship between Git and hosting providers like GitHub 4.6.8 List major open-source projects using Git 4.6.9 Install and configure Git on desktop PC 4.6.10 Install and configure Git on Raspberry Pi 4.6.11 List and explain common terminology associated with distributed VCS's including: - Branch - Checkout - Clone - Stage - Commit - Conflict - Head - Repository - Initialise	

Solution Development			
	Integrated solution development For topics (5 to 11 (and 12)) the students should demonstrate the ability to **create code-based solutions for** various types of problems requiring them to utilise different code constructs or a combination of different constructs in the development of a computer program. The individual SOs should be covered in isolation as well as in an integrated manner towards larger solution development.		
Topic 5 **Introduction to a high-level programming language**  **For Python version 2 is NOT to be used, but version 3.X.**	5.1 **Introduction to the Python programming language** Content: • Creating a Hello World application using PyCharm • Python program structure • Syntax and semantics *Learning* Outcomes*:* *Students should be able to:* Demonstrate the development of code-based solutions comprising of code constructs mastered in level 2 5.1.1 Differentiate between lexis and syntax 5.1.2 Differentiate between an instruction and a comment 5.1.3 Define the term interpreter and elaborate on what function the interpreter serves. 5.1.4 Explain how an interpreter is different from a traditional compiler as used by C/C++ 5.1.5 Explain Pythons traditional runtime execution model (Source to Byte code to Runtime) 5.1.6 Explain why Python programs uses the .py extension. 5.1.7 Write the Python code to display output “Hello World” using a basic print statement in the interactive Python prompt 5.1.8 Create a new project using the high-level IDE 5.1.9 Write the Python code to display output “Hello World” using a basic print statement in PyCharm. 5.1.10 List and explain two types of comments allowed by the Python interpreter. 5.1.11 Explain the term block-structured language and how that applies to Python	5.1 **Creating a Hello World application** Content: • Geany IDE • Compiling • Building • Running *Learning* Outcomes*:* *Students should be able to:* 5.1.1 Create a new project using an IDE 5.1.2 Create a new file with extension CPP 5.1.3 Explain why C++ source files use the .cpp extension and C source files uses the .c extension. 5.1.4 Write the C/C++ code to display output “Hello World”. 5.1.5 Compile a C/C++ application using an IDE 5.1.6 Explain what type of file is produced by the C/C++ compilation process 5.1.7 Show or hide the IDE message window 5.1.8 Investigate the IDE message window to determine if compilation succeeded 5.1.9 Investigate the IDE message window to locate line numbers that contain errors 5.1.10 Build a C/C++ application using an IDE 5.1.11 Explain what file type is produced by the C/C++ build process 5.1.12 Run a C/C++ application using the IDE	 **For Python version 2 is NOT to be used, but version 3.X.**  For level 3 C++ input and output can be illusterated using cin and cout.

	5.1.12 Discuss how a block of code is defined in Python using whitespace indentation. 5.1.13 Explain how the whitespace indentation in Python is different to the braces {} as used by other programming languages. 5.1.14 Investigate the IDE message window to locate line numbers that contain error.	**5.2 C++ application structure** Content: • C/C++ fundamentals • Comments 5.2.1 List and explain the different parts of a simple C/C++ application 5.2.1.1 Headers 5.2.1.2 Name space 5.2.1.3 Main 5.2.1.4 return 5.2.2 Explain case sensitivity as it applies to C/C++ 5.2.3 Explain the term free-format language and how that applies to C/C++ 5.2.4 Explain the term block-structured language and how that applies to C/C++ 5.2.5 Explain what a comment is used for in the C/C++ language 5.2.6 Explain how the compiler will treat a comment 5.2.7 List and explain two types of comments allowed by the C/C++ compiler 5.2.8 Discuss the importance of adding comments to a C/C++ application	
Topic 6 **Data Types, Variables and Output** **LO 5.1.22 of the Robotics Curriculum** 	**6.1. Review generic concepts from Topics 3, 5 and introductory problem-solving concepts mastered with Scratch (This LO is used for reinforcement and ALSO link to LO 5.1.22 of the Robotics Curriculum) The relationships between the concepts coved should be highlighted i.e., Scratch and Python.** Content: • Working, creating and saving a python application • Open an existing application • Basic constructs • Problem solving principles **6.2. Python Data Types** Content: • Operators • Operator Precedence	**6.1. Data types in C/C++** Content: • Data Types in C/C++ • Arithmetic Operations • Variables *Learning* Outcomes: *Students should be able to* 6.1.1 Data Types in C/C++ 6.1.2 Define the term data type 6.1.3 Define the term literal value 6.1.4 Name and explain two numerical data types used in C/C++ 6.1.5 Explain the difference between Integer data types and floating-point data types 6.1.6 List the different integer data types	6.1 **Review generic concepts from Level 1 and 2** Content: • Data Types (General concepts only) • Arithmetic Operations • Variables • Multiple declarations • Arithmetic operations • Numbering systems • Escape characters and New Lines • Basic file output

	• Arithmetic expressions 6.2.1 Explain the Integer, Float, Boolean, String data types in terms of: 6.2.1.1 What the type can consist of 6.2.1.2 What the maximum and minimum value is 6.2.1.3 What operations is allowed on the type 6.2.1.4 Example literal values allowed for the type 6.2.1.5 Example literal values not allowed for the type 6.2.1.6 Whether the type is signed or unsigned **6.3. Arithmetic Operations** Content: • Operators • Operator Precedence • Arithmetic expressions *Learning Outcomes:* *Students should be able to:* 6.3.1 List the different arithmetic operators 6.3.2 Define the term binary operator 6.3.3 Define the term operand 6.3.4 List and explain (with examples) what a binary arithmetic expression consists of 6.3.5 Write Python code that uses print() to display the value of different arithmetic expressions (combined with string literal for context) 6.3.6 List and explain the data type produced based on the data type of the operands involved 6.3.7 List and explain the rules involved when creating complex arithmetic expressions containing multiple operands 6.3.8 List and describe the three levels of precedence 6.3.9 Write Python code that uses Print() to display the value of different complex arithmetic expressions (combined with string literal for context)	6.1.7 List the different floating-point data types 6.1.8 Explain the float, double data types in terms of: - What the type can consist of - What the maximum and minimum value can be for the type - What operations is allowed on the type - Example literal values allowed for the type - Example literal values not allowed for the type - Whether the type is signed or unsigned - Number of bytes memory used to store type value 6.1.9 Explain the int, char, bool, float double data types in terms of: - What the type can consist of - What the maximum and minimum value is - What operations is allowed on the type - Example literal values allowed for the type - Example literal values not allowed for the type - Whether the type is signed or unsigned - Number of bytes memory used to store type value 6.1.10 Use variables as part of a solution - Define the term variable - Explain how variables are used in C/C++ applications - Identify and list C/C++ key words - List and describe the syntax rules for naming variables - Write C/C++ code to create variables of different types - Write C/C++ code to assign literal values to variables of different types - Write C/C++ code store the result of an arithmetic expression (simple and complex) in a variable - Write C/C++ code to print the value associated with a variable (combined with string literal for context) - Write C/C++ code store the result of an arithmetic expression that contains variables on both sides of the operand (simple and complex) in a variable - Write C/C++ code to assign the value of one variable to a second variable	

	6.4. Number systems Content: • Decimal system • Binary system • Conversions *Learning* Outcomes*:* *Students should be able to* 6.4.1 Define the term positional number system 6.4.2 Indicate the base of a number using the base subscript 6.4.3 Define and explain the base of a decimal number 6.4.4 Rewrite a decimal number as the sum of its base 6.4.5 Define and explain the base of a binary number 6.4.6 Convert a binary number to a decimal number 6.4.7 Convert a decimal number to a binary number 6.4.8 Define and explain the base of a hexadecimal number 6.4.9 Explain how computers use hexadecimal numbers 6.4.10 List the hexadecimal numbers 0-9 A-F and their decimal equivalents 6.4.11 Convert a decimal number to a hexadecimal number **6.5. Working with characters and strings** Content: • Declaring and using character data types • Declaring and using string data types 6.5.1 List the different types of data associated with characters and strings 6.5.2 Initialise data as a character 6.5.3 Initialise data as a string 6.5.4 Manipulate character and string data for output purposes 6.5.5 Use common built-in functions to manipulate strings and characters Range: - find - format - index	- Explain the difference between variable declaration and variable initialisation - Write C/C++ code to initialise variables of different types - Write C/C++ code to increase or decrease the value of a variable (accumulation statements) - Write C/C++ code to increase or decrease the value of a variable with 1 using the increment operator - Write C/C++ code to print the value associated with a variable (combined with string literal for context) **6.2. Arithmetic Operations** Content: • Operators • Operator Precedence • Arithmetic expressions *Learning* Outcomes*:* *Students should be able to:* 6.2.1 List the different arithmetic operators 6.2.2 Define the term binary operator 6.2.3 Define the term operand 6.2.4 List and explain (with examples) what a binary arithmetic expression consists of 6.2.5 Write C/C++ code that uses printf or cout to display the value of different arithmetic expressions (combined with string literal for context) 6.2.6 List and explain the data type produced based on the data type of the operands involved 6.2.7 List and explain the rules involved when creating complex arithmetic expressions containing multiple operands 6.2.8 List and describe the three levels of precedence 6.2.9 Write C++ code that uses printf or cout to display the value of different complex arithmetic expressions (combined with string literal for context)	

	- isalpha - isascii - isdigit - lower - split 6.5.6 Concatenate strings 6.5.7 Convert a string to another datatype *Learning* Outcomes*:* *Students should be able to* **6.6. Basic File Output** Content: • Writing to text files *Learning* Outcomes*:* *Students should be able to* 6.6.1 Explain the purpose of each parameter used in the open() function to ready a file for writing using the wt mode. 6.6.2 Write Python code that opens a text file for writing, 6.6.3 Write Python that will use the file object to write to a text file 6.6.4 Write Python code to ensure a file was successfully opened and display appropriate error message if not the case 6.6.5 Write Python code to close a file	**6.3. Multiple declarations** Content: • Single statement declarations *Learning* Outcomes*:* *Students should be able to:* 6.3.1 Identify or correct the general form for multiple variable declarations 6.3.2 Explain why multiple declarations should be of the same type 6.3.3 Write C/C++ code that declares multiple variables in a single line **6.4. Escape characters and New Lines** Content: • Escape characters 6.4.1 Explain what meaning the backslash character will have in C/C++ 6.4.2 Define the term escape character 6.4.3 Define the term escape sequence 6.4.4 Write C/C++ code that uses printf or cout to display/use variable (combined with string literal and variable values for context): - Backslash - Question mark - Single and Double quotation - Horizontal tab - Alert **6.5. Formatted Output** Content: • Field with manipulators *Learning* Outcomes*:* *Students should be able to* 6.5.1 Give examples of where output in an application could be formatted to be more attractive to the user 6.5.2 Explain what field with manipulators are used for	

		6.5.3 Write C/C++ code that uses printf and formatting strings to: - Set the field width - Set the default leading fill character - Left justify numbers - Right justify numbers - Set floating point precision	

Topic 7 **Math, Interactive Input, Constants and Errors**	**7.1. Python Keyboard Input** Content: • input() function • Validation concepts *Learning Outcomes:* *Students should be able to* 7.1.1 Explain the process that happens when the computer encounters the input() statement 7.1.2 Use the Python input() function to read user input (numeric) and store the result in a variable 7.1.3 Use Python input() function to read user input (single character) and store the result in a variable 7.1.4 Define the term validation 7.1.5 Explain why user input validation is important 7.1.6 Convert user input or other values to Integer using the int() function. 7.1.7 Convert user input or other values to Float using the float() function. 7.1.8 Convert Integer and Float to String using the str() function. **7.2. Python Mathematical Library functions** Content: • Math Library • Preprogramed functions Learning Outcomes: Students should be able to	**7.1. C/C++ Keyboard Input** Content: • scanf function • cin object • Validation concepts *Learning* Outcomes*:* *Students should be able to* 7.1.1 Explain the process that happens when the computer encounters the scanf or cin statement 7.1.2 Use the C/C++ scanf function or cin to read user input (numeric) and store the result in a variable 7.1.3 Use the C/C++ scanf function or cin object to read user input (single character) and store the result in a variable 7.1.4 Define the term validation 7.1.5 Explain why user input validation is important **7.2. C/C++ Errors** Content: • Syntax errors • Logic Errors *Learning* Outcomes*:* *Students should be able to* 7.2.1 Define the term syntax 7.2.2 Define the term syntax error 7.2.3 Identify and correct syntax errors in C/C++ code 7.2.4 Explain the relationship between syntax and compile time errors	**7.1. Review generic concepts from Level 2 and 3** Content: • Validation • Symbolic Constants **7.2. Basic File input** Content: • Text file reading *Learning* Outcomes*:* *Students should be able to* 7.2.1 Explain the purpose of each parameter used in the open() function to ready a text file for reading with the rt mode. 7.2.2 Write Python code that opens a text file for reading 7.2.3 Write Python code to ensure a file was successfully opened and display appropriate error message if not the case 7.2.4 Write Python code that will read all lines from a file one by one. 7.2.5 Write Python code to close a file

	7.2.1 Explain the concept of a preprogramed (library) module in Python 7.2.2 Explain the need for preprogramed math functions 7.2.3 Write Python code to import the math module 7.2.4 Write Python code that makes use of the following common mathematical module functions - fabs(..) - pow(…) - sqrt(…) 7.2.5 List the mathematical functions covered and the return type they will produce 7.2.6 Write Python code containing complex mathematical expressions and math functions	7.2.5 Define the term typographical error 7.2.6 Explain when a typographical error is a syntax error 7.2.7 Define the term logic error 7.2.8 List and explain three common side effects of logic errors **7.3. Math library** Content: • Errors in code • Math Library • Preprogramed functions *Learning* Outcomes*:* *Students should be able to* 7.3.1 Math Library 7.3.1.1 Explain the concept of a preprogramed function 7.3.1.2 Explain the need for preprogramed math functions 7.3.1.3 Explain the purpose of the math.h header file 7.3.1.4 Write C/C++ code that includes the header file math.h 7.3.2 C++ Errors 7.3.2.1 Define the term syntax error 7.3.2.2 Identify and correct syntax errors in C/C++ code 7.3.2.3 Define the term logic error 7.3.2.4 List and explain three common side effects of logic errors 7.3.3 Write C++ code that makes use of the following common mathematical functions 7.3.3.1 abs(n) 7.3.3.2 pow(n,n2) 7.3.3.3 sqrt(n) 7.3.3.4 sin(n) 7.3.3.5 cos(n,n2) 7.3.3.6 tan(n) 7.3.4 List the mathematical functions covered and the return type they will produce 7.3.5 Write C/C++ code containing complex mathematical expressions and math functions	

		7.4. Validation Content: • Validating Numeric input • Validation concepts *Learning* Outcomes*:* *Students should be able to* 7.4.1 Define the term validation 7.4.2 Explain why user input validation is important 7.4.3 Write C/C++ code that will validate numeric input ensuring the input is within a pre-defined range 7.4.4 Write C/C++ code that will validate character input ensuring the expected input is one of 3 (max) characters **7.5. Symbolic Constants** Content: • Constants *Learning* Outcomes*:* *Students should be able to* 7.5.1 Define the concept of a constant value 7.5.2 List common constants found in everyday life 7.5.3 Define the term magic numbers as used by programmers 7.5.4 Explain the advantage of using a constant for magic numbers when programming 7.5.5 Define the term symbolic name 7.5.6 Write C/C++ code that uses the const declaration qualifier to mark a value as constant	

Topic 8 **Selection control structure**	8.1 **Review generic concepts from Topics 3, 5 and introductory problem-solving concepts mastered with Scratch** **(This LO is used for reinforcement and ALSO link to LO 5.1.22 of the Robotics Curriculum) The relationships between the concepts coved should be highlighted i.e., Scratch and Python.** Content: • Relational expressions • Selection Statement • Logical Operators  8.2 **Conditional tests and logic operators** Content: • Relational operators • Logical operators Learning Outcomes: *Students should be able to* 8.2.1 List and explain the different Python relational operators (== != > >= < <=) 8.2.2 Write Python code that will save the result of relation expression that contains both or a mix of numeric variables and literals. 8.2.3 Explain the purpose of the different Python logic operators AND, OR and NOT 8.2.4 Write Python code that will save the result of relation expression that contains both or a mix of numeric variables, character variables and literals and multiple (Max 3) logical operators 8.3 **Selection Statement** Content: • if statement • else statement • nested if statement • if elif else chain *Learning* Outcomes: *Students should be able to*	8.1 **Relational expressions** Content: • Relational operators *Learning* Outcomes*:* *Students should be able to* 8.1.1 Define the term relational expression 8.1.2 Explain the anatomy of a simple relational expression 8.1.3 List and explain the different C/C++ relational operators 8.1.4 Identify valid relational expressions 8.1.5 Identify invalid relational expressions 8.1.6 Explain the numeric result that is generated from a relational expression 8.1.7 Write C/C++ code that will save the result of relation expression that contains both or a mix of numeric variables and literals in an int variable 8.1.8 Write C/C++ code that will save the result of relation expression that contains both or a mix of numeric variables and literals in a Boolean variable 8.1.9 Explain how the integer result produced by a relation expression relates to bool true/false 8.1.10 Write C/C++ code that will save the result of relation expression that contains both or a mix of character variables and literals in an int and bool variable 8.2 **Selection Statement** Content: • If statement • Else statement *Learning* Outcomes*:* *Students should be able to* 8.2.1 Explain the purpose of the if statement in C/C++ 8.2.2 Explain the purpose of the else statement in C/C++ 8.2.3 Determine the program flow when an if statement is encountered 8.2.4 Identify or correct the general form for an if-else statement 8.2.5 Write C/C++ code that will use relational expressions in if-else statements (Max 2 two nested levels only)	8.1 **Review generic concepts from Level 2 and 3** Content: • Relational expressions • Selection Statement • Logical Operators 8.2 Problem solving using selection control structures Content: • Relational operators • Logical operators Learning Outcomes: *Students should be able to* 8.2.5 List and explain the different Python relational operators (== != > >= < <=) 8.2.6 Write Python code that will save the result of relation expression that contains both or a mix of numeric variables and literals. 8.2.7 Explain the purpose of the different Python logic operators AND, OR and NOT 8.2.8 Write Python code that will save the result of relation expression that contains both or a mix of numeric variables, character variables and literals and multiple (Max 3) logical operators 8.3 **Selection Statement** Content: • if statement • else statement • nested if statement • if elif else chain *Learning* Outcomes: *Students should be able to* 8.3.7 Identify or correct the general form for a Python if-else statement 8.3.8 Write Python code that will use relational expressions and logic operators in if-else statements

	8.3.1 Identify or correct the general form for a Python if-else statement 8.3.2 Write Python code that will use relational expressions and logic operators in if-else statements 8.3.3 Identify or correct the general form for a nested if-else statement 8.3.4 Write Python code that will use relational expressions in nested if-else statements (Max 3 levels) 8.3.5 Identify or correct the general form for a Python if else chain 8.3.6 Write Python code that will use an if else chain (Max 5 chain length)	8.2.6 Write C/C++ code that will use relational expressions in if-else statements with compounded content (Max 2 two nested levels only) 8.2.7 Explore the concept of block scope when writing if-else statements with compounded content 8.3 **Logical Operators** Content: • Using logical operators *Learning* Outcomes*:* *Students should be able to* 8.3.1 Define the term logic operator 8.3.2 List the different logic operators 8.3.3 Explain how each logic operator will influence the result of an expression 8.3.4 Write C/C++ code that will save the result of relation expression that contains both or a mix of numeric variables, character variables and literals and a single logical operator in an int variable 8.3.5 Write C/C++ code that will save the result of relation expression that contains both or a mix of numeric variables, character variables and literals and a single logical operator in a bool variable 8.3.6 Write C/C++ code that will save the result of relation expression that contains both or a mix of numeric variables, character variables and literals and multiple (Max 3) logical operator in a bool variable 8.3.7 Write C/C++ code that will save the result of relation expression that contains both or a mix of numeric variables, character variables and literals and multiple (Max 3) logical operator in an int variable 8.4 **Selection Statements** *Learning* Outcomes*:* *Students should be able to* Content: • Nested If statement • Else statement • Logical operator based if statement	8.3.9 Identify or correct the general form for a nested if-else statement 8.3.10 Write Python code that will use relational expressions in nested if-else statements (Max 3 levels) 8.3.11 Identify or correct the general form for a Python if else chain 8.3.12 Write Python code that will use an if else chain (Max 5 chain length)

		8.4.1 Define the term nested if statement 8.4.2 Determine the application flow when a nested if statement is encountered 8.4.3 Identify or correct the general form for a nested if-else statement 8.4.4 Write C/C++ code that will use relational expressions in nested if-else statements (Max 3 levels) 8.4.5 Write C/C++ code that will use relational expressions in nested if-else statements with compounded content (Max 3 levels) 8.4.6 Write C/C++ code that will use relational expression containing logic operators in nested if-else statements (Max 3 levels, Max 3 logic operators per level) 8.4.7 Write C/C++ code that will use relational expressions containing logic operator in nested if-else statements with compounded content (Max 3 levels, Max 3 logic operators per level)	

Topic 9 **Repetition control structure**	9.1 **Review generic concepts from Topics 3, 5 and introductory problem-solving concepts mastered with Scratch** **(This LO is used for reinforcement and ALSO link to LO 5.1.22 of the Robotics Curriculum) The relationships between the concepts coved should be highlighted i.e., Scratch and Python.**  Content: **While Repetition control** **Logic operators in While loop** 9.2 **While Repetition control** *Learning* Outcomes*:* *Students should be able to* 9.2.1 Write Python code that will use relational expressions and logic operators containing logic operators in while (Max 1 level, Max 1 logic operators per level) 9.2.2 Write Python code that will use relational expressions containing logic operator in while loop with	9.1 **While Repetition control** Content: • While loop • Loop evaluation • Break / Continue *Learning* Outcomes*:* *Students should be able to* 9.1.1 Define the term pre-test loop and entrance-controlled loop 9.1.2 Define the term fixed-count loop 9.1.3 Define the term variable condition loop 9.1.4 Explain the purpose of the while statement in C/C++ 9.1.5 Determine the application flow when a while statement is encountered 9.1.6 Identify or correct the general form for a while statement 9.1.7 Write C/C++ code that will use relational expressions in while statement. (Max 2 levels only) 9.1.8 Write C/C++ code that will use relational expressions in while statements with compounded content (Max 2 two nested levels only)	**9.1. Review generic concepts from Level 2 and 3** Content: **While Repetition control** **Logic operators in While loop** **9.2. Nested loops** *Learning* Outcomes*:* *Students should be able to* Content: • Solve problems using nested loops 9.2.1 Define the term nested loop 9.2.2 Define the terms inner and outer loops 9.2.3 Write Python code that will nest identical type loops (Max 3 levels) 9.2.4 Write Python code that will nest different type loops (Max 3 levels) 9.2.5 Determine the application flow when nested loop statement is encountered

	compounded content (Max 1 level, Max 1 logic operators per level) 9.2.3 Explain what a break statement is used for in Python while loop. 9.2.4 Explain what the pass statement is used for in Python while loop. 9.2.5 Explain what the continue statement is used for in Python while loop. in Python while loop. 9.2.6 Write Python code that incorporates the break, continue and pass statement. 9.3. **For Repetition control** Content: • For loop Learning Outcomes: *Students should be able to* 9.3.1 Identify or correct the general form for a for statement using the range function 9.3.2 Explain the purpose of each part in the for-loop initialisation using the range function 9.3.3 Use the break statement to end an infinite for loop based on the loop reaching a condition 9.3.4 Write Python code that will use a for loop with a predetermined number of loops as defined by the range function. 9.3.5 Write Python code that will use a for loop with a non-sequential counter variable as defined by the range function. 9.4 **Nested loops** *Learning* Outcomes*:* *Students should be able to* Content: • Nested loops 9.4.1 Define the term nested loop 9.4.2 Define the terms inner and outer loops	9.1.9 Explore the concept of block scope when writing while statements with compounded content 9.1.10 Explain the difference between fixed-count loop and variable condition loop 9.1.11 Write C/C++ code that will use relational expressions in while statements with interactive input (Max 2 two nested levels only) 9.1.12 Explain what a break statement is used for 9.1.13 Explain what the continue statement is used for 9.1.14 Write C/C++ code that will use relational expressions in while statements making use of the break statement 9.1.15 Write C/C++ code that will use relational expressions in while statements making use of the continue statement 9.2. **While Repetition control** Content: • While Repetition control *Learning* Outcomes*:* *Students should be able to* 9.2.1 While Repetition control 9.2.1.1 Define the term pre-test loop and entrance-controlled loop 9.2.1.2 Define the term fixed-count loop 9.2.1.3 Define the term variable condition loop 9.2.1.4 Explain the purpose of the while statement in C++ 9.2.1.5 Determine the application flow when a while statement is encountered 9.2.1.6 Identify or correct the general form for a while statement 9.3. **Logic operators in While loop** Content: • While loop • Logic operators *Learning* Outcomes*:* *Students should be able to*	

	9.4.3 Write Python code that will nest identical type loops (Max 2 levels) 9.4.4 Write Python code that will nest different type loops (Max 2 levels) 9.4.5 Determine the application flow when nested loop statement is encountered	9.3.1 Write C/C++ code that will use relational expression containing logic operators in while (Max 3 level, Max 3 logic operators per level) 9.3.2 Write C/C++ code that will use relational expressions containing logic operator in while loop with compounded content (Max 3 level, Max32 logic operators per level) 9.4. **Do While loop** Content: • Do While loop • Logic operators *Learning* Outcomes*:* *Students should be able to* 9.4.1 Define the term post-test loop and exit controlled loop 9.4.2 Determine the application flow when a do while statement is encountered 9.4.3 Identify or correct the general form for a do while statement 9.4.4 Define the term sentinel 9.4.5 Explain where and for what reason a sentinel is used 9.4.6 Write C/C++ code that will use relational expression containing logic operators in a do while loop (Max31 level, Max 3 logic operators per level) 9.4.7 Write C/C++ code that will use relational expressions containing logic operator in a do while loop with compounded content (Max 3 level, Max 3 logic operators per level) 9.4.8 Write C/C++ code that will use relational expression containing logic operators in a sentinel controlled do while loop (Max 3 level, Max 3 logic operators per level) 9.4.9 Write C/C++ code that will use relational expressions containing logic operator in a sentinel controlled do while loop with compounded content (Max 3 level, Max 3 logic operators per level)	

		9.5. **The for loop** *Learning* Outcomes*:* *Students should be able to* Content: • For loop *Learning* Outcomes*:* *Students should be able to* 9.5.1 Define the term variable condition loop 9.5.2 Explain how the, for statement differs from the while 9.5.3 Explain how the, for statement is similar to the while 9.5.4 Identify or correct the general form for a for statement 9.5.5 Explain the purpose of each part in the for-loop initialisation 9.5.6 Explain what effect an empty semicolon will have on the for loop 9.5.7 Explain the term infinite loop 9.5.8 Write C++ code to create an infinite for loop 9.5.9 Use the break statement to end an infinite for loop based on the loop reaching a condition 9.5.10 Determine the application flow when a for statement is encountered 9.5.11 Write C++ code that will use a for loop with a predetermined amount of loops 9.5.12 Write C++ code that will use a for loop with a non-sequential counter variable 9.6. **Nested loops** *Learning* Outcomes*:* *Students should be able to* Content: • Nested loops 9.6.1 Define the term nested loop 9.6.2 Define the terms inner and outer loops 9.6.3 Write C++ code that will nest identical type loops (Max 3 levels) 9.6.4 Write C++ code that will nest different type loops (Max 3 levels) 9.6.5 Determine the application flow when nested loop statement is encountered	

		9.7. **Program flow and control** *Learning* Outcomes*:* *Students should be able to* Content: • Break and continue statements 9.7.1 Explain what a break statement is used for 9.7.2 Explain what the continue statement is used for 9.7.3 Implement a break or continue statement as required as part of a code solution 9.8. **Program flow and control** *Learning* Outcomes*:* *Students should be able to* Content: • Creating C/C++ solutions 9.8.1 Plan, Design, create and code a C++ solution incorporating various structures and nested strictest o solve a given problem.	

Topic 10 Modularisation and Functions	10.1 **Modularisation and functions** Content: • Variable Scope *Learning* Outcomes*:* *Students should be able to* 10.1.1 Define the term scope 10.1.2 Differentiate between the use of local, enclosing, global scope, and built-in variable scope in python. 10.2 **Simple Python Functions** Content: • void non parameter functions	**10.2. Variables and scope** Content: • Variable Scope *Learning* Outcomes*:* *Students should be able to* 10.2.1 Variable Scope 10.2.1.1 Define the term scope 10.2.1.2 Define the term local scope 10.2.1.3 Define the term global scope 10.2.2 Write C/C++ code that will create a global variable 10.2.3 Write C/C++ code that will modify the value of a global variable in main, a void function and main block scope or function block scope	10.1 **Review generic concepts from Level 2 and 3** Content: • Variable Scope 10.2 **Revisit concepts covered from Level 2 and 3** Content: • Modularisation • void non parameter functions • Calling functions

	Learning Outcomes*:* *Students should be able to* 10.2.1 List the two parts that most Python functions will consist of (def and ():) 10.2.2 Identify or correct the general form for non-parameter Python function 10.2.3 Write Python code that will create a non-parameter function that does not return a value 10.2.4 Write Python code that will call a non-parameter function that does not return a value 10.2.5 Write Python code that will use one or more non-parameter function(s) to modify a global variable	**10.3. Functions** Content: • Functions • Value returning functions • Function parameters (by value and by reference) • Solution development using function *Learning* Outcomes*:* *Students should be able to* 10.2.1 Define the term modularisation 10.2.2 Explain what a function is used for in programming 10.2.3 Explain the difference between a called function and calling function 10.2.4 Explain the purpose of the function prototype 10.2.5 Explain when a function prototype is needed 10.2.6 List the two parts that every C/C++ function will consist of 10.2.7 Identify or correct the general form for void non parameter function header 10.2.8 Identify or correct the general form for void non parameter function 10.2.9 Write C++ code that will create a void non parameter function prototype 10.2.10 Write C++ code that will create a void non parameter function implementation 10.2.11 Write C++ code that will call a void non parameter function from the main function 10.2.12 Write C++ code that will use one or more void non parameter function to modify a global variable 10.2.13 Identify or correct the general form for a value returning non parameter function header 10.2.14 Identify or correct the general form for a value returning non parameter function 10.2.15 Write C/C++ code that will create a value returning non parameter function prototype 10.2.16 Write C/C++ code that will create a value returning non parameter function implementation 10.2.17 Write C/C++ code that will call a value returning non parameter function from the main function 10.2.18 Write C/C++ code that will call a value returning non parameter function from other self-defined functions	10.3 **Advanced Python Functions** Content: • Functions • Function parameters *Learning* Outcomes*:* *Students should be able to* 10.3.1 Explain the purpose of parameters in Python functions 10.3.2 Explain how modifying a parameter value will affect other variables 10.3.3 Identify or correct the general form for a value returning parameter receiving function 10.3.4 Write Python code that will create a value returning parameter receiving function 10.3.5 Write Python code that will assign default values to function parameters 10.3.6 Write Python code that will call a value returning parameter receiving function 10.3.7 Write Python code that will call a value returning parameter receiving function from other self-defined functions 10.4 **Random numbers** Content: • Random number generation • Random number range *Learning* Outcomes*:* *Students should be able to* 10.4.1 Write Python code to generate a random number between the values n and *n (n type Number)* 10.4.2 Write Python code to generate a random number of random numbers

		10.4. Random numbers Content: • Random number generation *Learning* Outcomes*:* *Students should be able to* 10.4.1 Define the term random numbers 10.4.2 Define the term pseudo random numbers 10.4.3 Explain why generating truly random numbers is difficult 10.4.4 List the two general purpose functions for generating random numbers 10.4.5 Explain what the constant RAND_MAX is used for 10.4.6 Name the header file where the RAND_MAX value is defined 10.4.7 Determine the RAND_MAX value of the current compiler being used 10.4.8 Explain the purpose of the srand () function 10.4.9 Explain the purpose of the rand () function 10.4.10 Write C/C++ code to generate a random number between the values 0.0 and 1.0 10.4.11 Write C/C++ code to generate a random number between the values 0.0 and 1.0 10.4.12 Write C/C++ code to generate a random number between the values 0 and n (n type int) 10.4.13 Write C/C++ code to generate a random number between the values 1 and n (n type int)	

Topic 11 **Arrays and lists**	11.1 **One dimensional arrays and basic lists** Content: • Data structures • Arrays (1 Dimensional) • Lists *Learning* Outcomes*:* *Students should be able to* 11.1. Define a data structure 11.2. Differentiate between a list, array, and dictionary 11.3. Define a list and an array 11.4. Use a list in a program 11.5. Use an array in a program 11.6. Perform basic operations on an array Range - Traversing - Replace a value / elements - Counting value / elements - Sum - Find / calculate the mean - Find / calculate the mode - Find / calculate the average - Swopping to elements - Searching 11.7. Perform basic operations on a list Range - Traversing - Replace a value / elements - Counting elements / values - Sum - Find / calculate the mean - Find / calculate the mode - Find / calculate Average - Swopping to elements - Searching 11.8. Implement the following list methods as part of a solution Range - append() - clear()	11.1 **One dimensional arrays** Content: • Array one dimensional *Learning* Outcomes*:* *Students should be able to* 11.1. Define the term atomic variable 11.2. Define the term scalar variable 11.3. Give examples of atomic/scalar variables 11.4. Define the term one dimensional array 11.5. Identify or correct the general syntax of an array declaration statement 11.6. Write C/C++ code to define a one-dimensional array with a constant number of items 11.7. Explain the term array element 11.8. Explain the term subscript 11.9. Identify the subscript for the first element in a n array 11.10. Write C/C++ code to output, modify or assign an array value using the correct subscript 11.11. Write C/C++ code to input an array value using the correct subscript 11.12. Write C/C++ code to initialise an array with a list of values 11.13. Implement an array as part of a software solution to a given problem	11.1 **Review generic concepts from Level 2** 11.2 **Basic Python List** Content: • Creating and modifying lists Learning Outcomes*:* *Students should be able to* 11.2.1 Identify or correct the general syntax to create a Python list. 11.2.2 Write Python code to output, modify or assign a list value using the correct subscript 11.2.3 Write Python code to add an element to a list using the append() function. 11.2.4 Write Python code to add an element at a specific position to a list using the insert() function. 11.2.5 Write Python code to remove an element at a specific position to a list using the del statement. 11.2.6 Write Python code to remove the last element in a list and assign that element to a variable using pop(). 11.2.7 Write Python code to remove an element from a list by value using the remove statement 11.2.8 Sort a Python list permanently and temporarily using the sort() and sorted() functions. 11.2.9 Find the length of a list using the len() function 11.3 **Advanced Python Lists** Content: • looping over lists • contrasting python list types *Learning* Outcomes*:* *Students should be able to* 11.3.1 Write Python code that will loop over every element in a list using the for in loop structure 11.3.2 Generate a python list using the range() function 11.3.3 Write python code to print a specific section of a list using a slice

	- copy() - count() - extend() - index() - insert() - pop() - remove() - reverse() - sort()		11.3.4 Write Python code to loop through the elements of a list that was narrowed down with a slice 11.3.5 Write Python code to copy an entire list using a slice 11.3.6 Compare and contrast a Python list and Python tuple. 11.3.7 Compare and contrast a Python list and Python set. 11.4 **Python dictionary** Content: • Python key value pairs • Python dictionary *Learning* Outcomes*:* *Students should be able to* 11.4.1 Explain the concept of a key-value pair in Python dictionary 11.4.2 Identify or correct the general syntax to create a Python dictionary 11.4.3 Write Python code to access the value associated with a key in a dictionary 11.4.4 Write Python code to add a new key value paid to a dictionary 11.4.5 Write Python code to modify the value associated with a key in a dictionary 11.4.6 Remove a key value pair from a dictionary using the del statement 11.4.7 Write Python code using for key, value syntax to loop over every value pair in a dictionary 11.4.8 Write Python code using a for loop to loop through all the keys in a dictionary 11.4.9 Write Python code using a for loop to loop through all the values in dictionary

Topic 12 Concepts of object orientation		**Note: Level 3** **Topic 12 is the only topic specific to C++ and does not apply to C** **12.1. Fundamentals of OOP** Content: OOP in concepts *Learning* Outcomes*:* *Students should be able to* 12.1.1 Define the term Object Orientated Programming 12.1.1 Define the term: abstract data type 12.1.1 Differentiate between methods and attributes 12.1.1 Define the term instantiation 12.1.1 Describe the purpose of a constructor 12.1.1 Give examples of C++ built in classes 12.1.1 Instantiate an object using an existing class or external class 12.1.1 External classes: 10.4.13.1 Write C++ code to include the header file for an externally developed class (not part of standard C++ library) 10.4.13.2 Write C++ code to create or initialise an object using externally developed class 10.4.13.3 Write C++ code to Interact with object created from an externally developed class **12.2. The string class** Content: String fundamentals *Learning* Outcomes*:* *Students should be able to* 12.2.1 Define the term string literal 12.2.1 Give examples of string literals 12.2.1 Explain the concept of storing a string as an array of characters 12.2.1 Define the term object as it applies to C++ classes 12.2.1 Explain what header file is needed to use string objects	**12.1 Fundamentals of OOP** Content: OOP in concepts *Learning* Outcomes*:* *Students should be able to* 12.5.1 Define the term Object Orientated Programming 12.5.1 Define the term: abstract data type 12.5.1 Differentiate between methods and attributes 12.5.1 Define the term Class, Class Variable, Data member, Instance, Object 12.5.1 Create a new class using the class statement 12.5.1 Add class methods and variable members to a Python class 12.5.1 Instantiate an object from a Python class 12.5.1 Access attributes through an Object of a Python class

		12.2.1 Write C++ code to create a string object initialised with a string literal 12.2.1 Write C++ code to output, modify or assign individual characters of a string object 12.2.1 List the basic methods and objects for the input and output of string variables 12.2.1 Write C++ code to input a string object value using cin 12.2.1 Write C++ code to input a string object value using getline(..) 12.2.1 Explain the difference between cin and getline(...) when used for string object input 12.2.1 Explain the phantom new line character problem 12.2.1 Apply code to eliminate the phantom new line character problem **12.3. String processing** Content: Manipulating and implementing strings *Learning* Outcomes*:* *Students should be able to* 12.3.1 List, Explain and Use as part of C++ solution the following string object processing methods 12.3.1.1 lengh () 12.3.1.2 at(index), 12.3.1.3 compare(str) 12.3.1.4 erase (ind, n) 12.3.1.5 find(str) 12.3.1.6 substr(ind,n) 12.3.1 Write C++ code that combines different string processing methods to solve string processing problems **12.4. External classes** Content: Interacting with externally developed classes *Learning* Outcomes*:* *Students should be able to* 12.4.1 Write C++ code to include the header file for an externally developed class (not part of standard C++ library)	

		12.4.1 Write C++ code to create or initialise an object using externally developed class 12.4.1 Write C++ code to interact with an object created from externally developed class 12.5. Using cout and cin Content: IO using cout and cin *Learning* Outcomes*:* *Students should be able to* 12.5.1 Explain the concept of a stream 12.5.1 Differentiate between cin and cout as input and output stream objects in C++ 12.5.1 Explain the purpose and use of the stream insertion and stream extraction operators 12.5.1 Use cin and cout for input and output purposes as part of a solution 12.5.1 Apply various formatting	
Generic Algorithms **SOME! Suggested generic programming algorithms to be covered across all levels where the required coding constructs have been covered.**	Determine smallest, largest value of more than two values Swapping values Determining aggregates: sum, average (without and with loops) Basic calculations: calculating area, volume, VAT and others Determine whether a number is even or uneven Determine whether a number is a factor of another number Determine whether a number is a prime number Lowest common multiple (LCM), greatest common divisor (GCD) Find a specified character in a string Use special characters such as a star to draw with code (simple geometrical shapes) Search using the linear search algorithm Sorting (bubble and selection sort) Printing a list of numbers in reverse Generating Fibonacci sequence Convert between different metric units. Determine the number of days, hours, minutes, seconds in a unit of time. Determine change required, return on investment, profit or loss on a transaction		

	Motivate the use of a specific algorithm BigO notation basics Relate, combine and integrate several code structures/constructs to devise ‘new’ algorithms/ adapting existing ones

Standardisation and considerations

For educational and training purposes careful consideration should be given towards selecting tools, components, hardware, and software across the curriculum for the different levels.
The motivation regarding the use of standardised tools and hardware resides in the following points as it helps ensure:

- fair teaching & learning and setting of fair examination papers, and
- stability/compatibility during examinations

In addition, other motivational factors include:

National exam paper setting

Problems that exam panels can experience if different tools/components/hardware and software are used with different features and configurations. The complexities to synchronise the different tools and versions when setting practical papers impact on content coverage, and cognitive demand and/or time required in answering questions.

Marking of exams

Not standardising implies that markers need to know each of the available components and if colleges do not standardise marking could become very difficult.

Migration of lecturers and students

Students and lecturers moving from one college to another could be faced with challenges if they do not have the necessary prerequisite knowledge or knowledge of the new tools or hardware being used.

Lecturer support

Good ICT lecturers are very scarce and subject support is very specialised. With standardised tools lecturer support is less troublesome. This is also true for lecturer training programmes.

Technical support

This is also made difficult in terms of license agreements, setup, advise, etc.

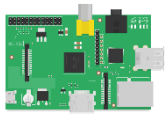
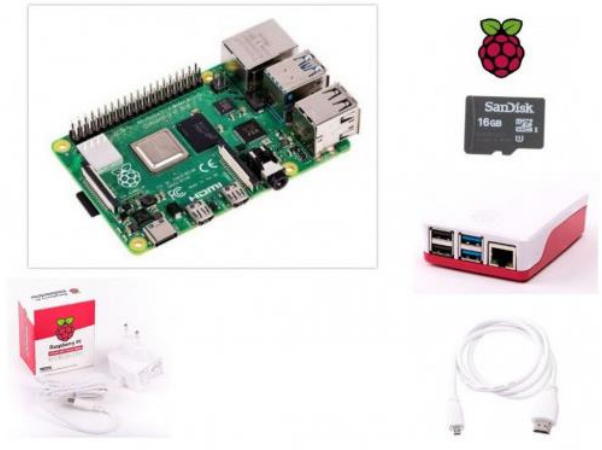
Motivating block based visual programming for introductory concepts

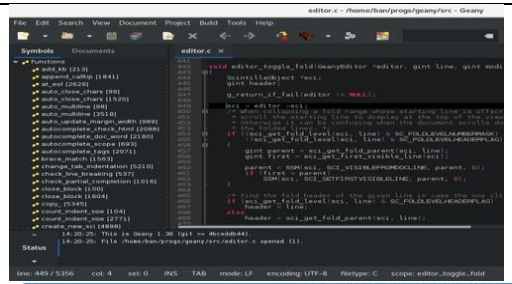
Various studies suggest and motivates the benefits of block-based visual programming to expose novices to the concept of programming, coding, and computational thinking (see, Asgedom, 2021 & Kong, & Wang, 2021).

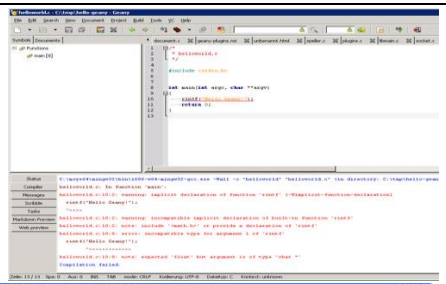
Scratch was created by Massachusetts Institute of Technology (MIT) and designed especially for ages 8 to 16. Scratch has a huge community with numerous resources that includes learning guides, teaching guides, lessons, videos, etc. Scratch offers a low floor (easy to get started, even with no experience of coding), wide walls (allows for all kinds of projects – from animations and games to science experiments) and high ceiling (the possibility to create complex projects) with a playful

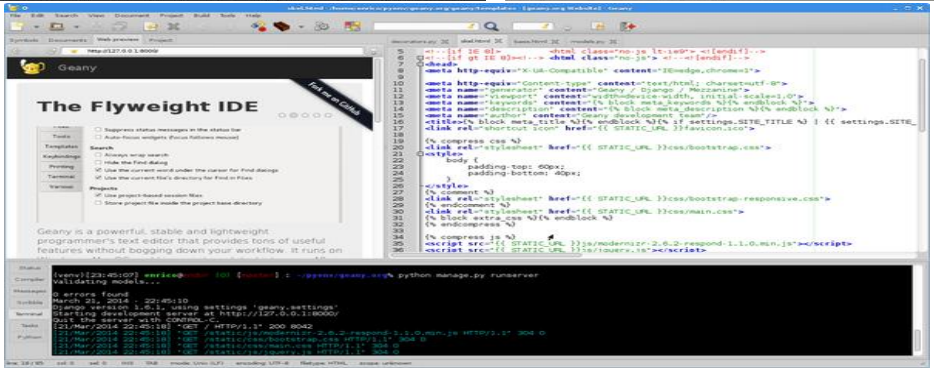
experience that teaches novices to code in an experimental fashion – it starts where the beginner is and allows for incremental development. It is therefore possible to use it throughout the curriculum, from Grade 1 to Grade 9. Scratch is a pathway to high level programming languages as it allows novices to develop coding constructs and to focus on logical thinking before focusing on syntax, which is one of the features of high-level programming languages (designed for industry) that makes them challenging. Numerous research projects (https://scratch.mit.edu/info/research/) suggest the value of Scratch to teach coding to novices, e.g. Meerbaum et al. (2010) found that a programming course that used Scratch, resulted in meaningful learning. Students were able to internalise programming concepts and showed improved cognitive achievement.
Scratch shipped with the Raspberry PI allows for the creation of code that enable GPIO programming on a concrete level. It also enables the student to be exposed to physical computing. Please see (https://projects.raspberrypi.org/en/projects/physical-computing-with-scratch/3)

Suggested hardware and software requirements for standardisation purposes

	L2 Basic principles of Computer programming and Computer literacy **L3 Introduction to Technical Programming** **L4 Technical Programming**
Hardware  	***Raspberry Pi – Latest available model***  ***Standard Desktop Computer*** I5 or better 4GB Ram 120GB HD
Software	***Geany IDE*** Geany is a powerful, stable and lightweight programmer's text editor that provides tons of useful features without bogging down your workflow. It runs on Windows, MacOS, Linux and the Rasberry Pi, is translated into over 40 languages, and has built-in support for more than 50 programming languages. One the primary reasons for Geany's existence is that there is a need for a decent, GUI, lightweight, cross-platform, flexible and powerful IDE/editor. Many editors meet one or more of these requirements but fall short on others. Geany is Good Ol' Fashioned Free Software. Using the GPL v2 license, Geany assures that not only can you customise and hack it, but that everyone benefits from changes the community makes. Geany has a vibrant community of professionals and hobbyists alike.







GNU tool chain

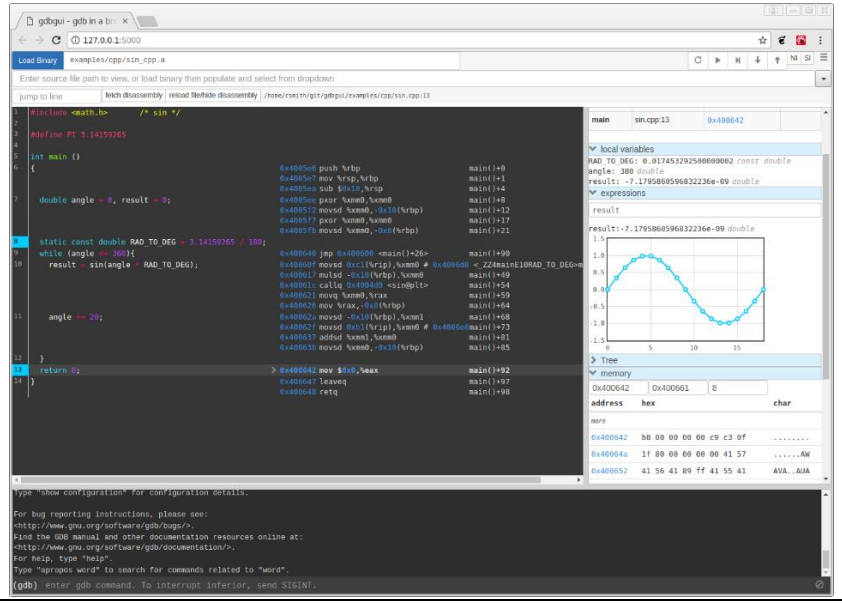
The GNU toolchain is a broad collection of programming tools produced by the GNU Project. These tools form a toolchain used for developing software applications and operating systems

MinGW

MinGW, a contraction of "Minimalist GNU for Windows", is a minimalist development environment for native Microsoft Windows applications.
MinGW provides a complete Open Source programming tool set which is suitable for the development of native MS-Windows applications, and which do not depend on any 3rd-party C-Runtime DLLs. (It does depend on a number of DLLs provided by Microsoft themselves, as components of the operating system; most notable among these is MSVCRT.DLL, the Microsoft C runtime library. Additionally, threaded applications must ship with a freely distributable thread support DLL, provided as part of MinGW itself).
MinGW compilers provide access to the functionality of the Microsoft C runtime and some language-specific runtimes. MinGW, being Minimalist, does not, and never will, attempt to provide a POSIX runtime environment for POSIX application deployment on MS-Windows. If you want POSIX application deployment on this platform, please consider Cygwin instead.
Primarily intended for use by developers working on the native MS-Windows platform, but also available for cross-hosted use, (see note below -- you may need to follow the "read more" link to see it), MinGW includes:

- A port of the GNU Compiler Collection (GCC), including C, C++, ADA and Fortran compilers;
- GNU Binutils for Windows (assembler, linker, archive manager)
- A command-line installer, with optional GUI front-end, (mingw-get) for MinGW and MSYS deployment on MS-Windows
- A GUI first-time setup tool (mingw-get-setup), to get you up and running with mingw

gdbGui

	gdbgui is a browser-based frontend to gdb, the gnu debugger. You can add breakpoints, view stack traces, and more in C, C++, Go, and Rust! It's perfect for beginners and experts. Simply run gdbgui from the terminal to start the gdbgui server, and a new tab will open in your browser. 

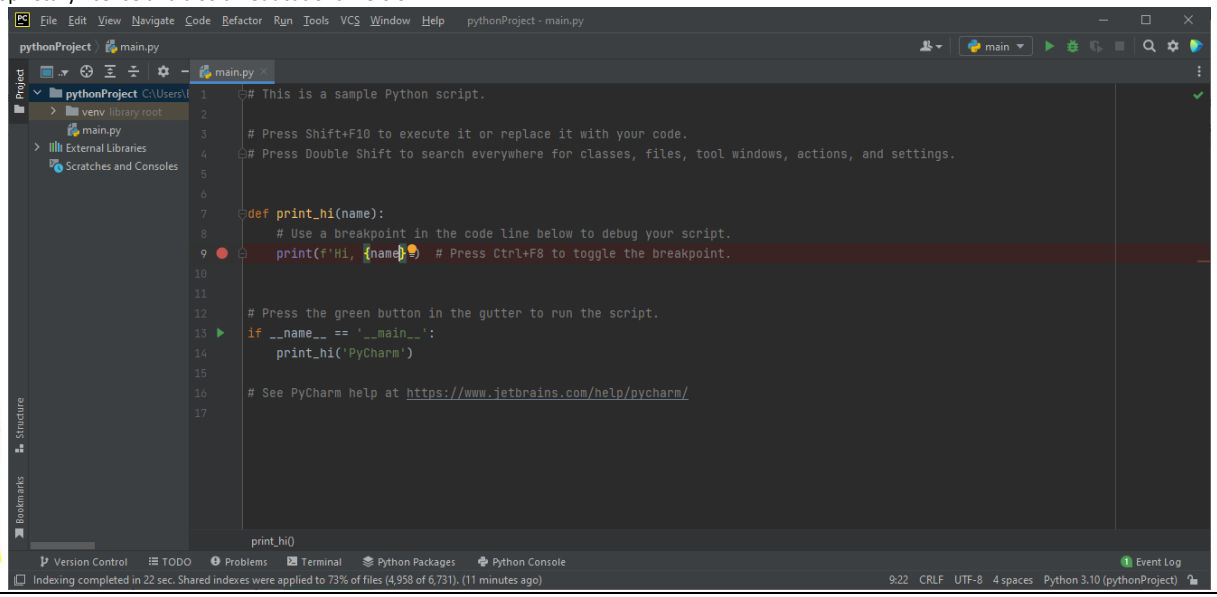
Python and PyCharm IDE

Python version 3.X is prescribed, and Version 2 is NOT to be USED!

Python is an interpreted high-level general-purpose programming language. Its design philosophy emphasizes code readability with its use of significant indentation. Its language constructs as well as its object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

PyCharm is an integrated development environment (IDE) used in computer programming, specifically for the Python programming language. It is developed by the Czech company JetBrains (formerly known as IntelliJ). It provides code analysis, a graphical debugger, an integrated unit tester, integration with version control systems (VCSes), and supports web development with Django as well as data science with Anaconda.

PyCharm is cross-platform, with Windows, macOS and Linux versions. The Community Edition is released under the Apache License,[7] and there is also Professional Edition with extra features – released under a proprietary license and also an educational version.



Cheat sheet – useful links and references

In no particular order and for insight and perusal

https://learn.sparkfun.com/tutorials/python-programming-tutorial-getting-started-with-the-raspberry-pi/programming-in-python

https://www.raspberrypi.org/blog/scratch-3-desktop-for-raspbian-on-raspberry-pi/

https://www.makeblock.com/coding-robot/208984.html

http://blog.ardublock.com/category/tutorial/

https://www.arduino.cc/en/software

https://www.pythonforthelab.com/blog/how-control-arduino-computer-using-python/

https://learn.sparkfun.com/ArduBlock

https://roboticsbackend.com/how-to-learn-python-3-with-raspberry-pi-from-scratch/

https://roboticsbackend.com/category/raspberry-pi/

https://pimylifeup.com/getting-started-with-python/

References

Abelson, H., Sussman, G.J. (1996). Structure and Interpretation of Computer Programs, 2nd edition. Series MIT Electrical Engineering and Computer Science.

Bell, T., Witten, I.H., Fellows, M. (1998). Computer Science Unplugged. Off-line activities and games for all ages. http://unplugged.canterbury.ac.nz

Bloem, J. et al., 2014. The fourth industrial revolution. Things Tighten. Available at: https://www.sogeti.com/globalassets/global/special/sogeti-things3en.pdf.

Anon, C++ Glossary - Step by Step Programming. Available at: https://sites.google.com/site/simplestjava/c-glossary [Accessed August 14, 2019].

Anon, gdbgui. Available at: https://www.gdbgui.com/ [Accessed August 14, 2019].

Asgedom, N., 2021. "Robotic and block-based programming with Text Interpreter for motivating novice programmers." Master's thesis, Itä-Suomen yliopisto, 2021.

Candido, R., (2019). Arduino With Python: How to Get Started Available at: https://realpython.com/arduino-python/

Corke, P., Haviland, J., (2021). *Not your grandmother's toolbox–the Robotics Toolbox reinvented for Python*. In IEEE International Conference on Robotics and Automation.

Ellen Frederick, D., (2016). Libraries, data and the fourth industrial revolution (Data Deluge Column). Library Hi Tech News, 33(5), pp.9–12.

Futurelearn.com (2021) Programming skills for robotics. Available at: https://www.futurelearn.com/info/courses/robotic-future/0/steps/29368

Ginat, D. (2006). On novices' local views of algorithmic characteristics. In: Mittermeir, R.T. (Ed.): ISSEP 2006, LNCS, 4226, 127–137.

Govender, I. (2006). Learning to Program, Learning to Teach Programming: Pre- and In-service Teachers' Experiences of an Object-oriented Language. University of South Africa.

Hromkovi˘c, J. (2006). Contributing to general education by teaching informatics. In: Mittermeir, R.T. (Ed.), ISSEP 2006, LNCS, 4226, 25–37.

Hymel, S., (2021) *Python Programming Tutorial: Getting Started with the Raspberry* Pi Available at: https://learn.sparkfun.com/tutorials/python-programming-tutorial-getting-started-with-the-raspberry-pi/programming-in-python

Joseph L. (2018) Fundamentals of C++ for Robotics Programming. In: Robot Operating System (ROS) for Absolute Beginners. Apress, Berkeley, CA

King, P, (n.d.), Simple electronics with GPIOZero, The Magpi magazine. Online available at: https://www.mclibre.org/descargar/docs/revistas/magpi-books/the-magpi-essentials-gpiozero-01-en-201608.pdf

Kong, S.C. & Wang, Y.Q., 2021. The influence of parental support and perceived usefulness on students' learning motivation and flow experience in visual programming: Investigation from a parent perspective. *British Journal of Educational Technology*

Mannila, L. (2007). Novices' progress in introductory programming courses. Informatics in Education, 6, 139– 152.

Anon, MinGW | Minimalist GNU for Windows. Available at: http://www.mingw.org/ [Accessed August 14, 2019].

Meerbaum-Salant, O., Armoni, M. & Ben-Ari, M. (moti), 2013. Learning computer science concepts with Scratch. Computer Science Education, 23(3), pp.239–264.

Mehla, A,. (2021) Python Language advantages and applications – GeeksforGeeks Available at: https://www.geeksforgeeks.org/python-language-advantages-applications/

Mulder, F. (2002). van BÈTA – naar DELTA-discipline (in English: Computer Science: from a BÈTA to a D E LTA s u b j e c t) . Informatica, Tinfon, 11, 48.

Owen-Hill, A. (2018), Python vs C++ vs C# vs MATLAB: Which Robot Language is Best? Available at: https://robodk.com/blog/robot-programming-language/

O'Hanlon, M., (2019) Scratch 3 Desktop for Raspbian on Raspberry Pi ? Available at: https://www.raspberrypi.org/blog/scratch-3-desktop-for-raspbian-on-raspberry-pi/

Papert, S. (1980). Mindstorms. Children, Computers and Powerful Ideas. Basic Books, Inc. Publishers, New Yo r k .

Practpoint, 2019. File:PHP - learn php programming online programming practice.jpg. Available at: https://commons.wikimedia.org/wiki/File:PHP_-_learn_php_programming_online_programming_practice.jpg [Accessed July 4, 2019].

RoboticsBackend.com (2021) When To Use Python vs C++ in Robotics? Available at: https://roboticsbackend.com/when-to-use-python-vs-c-in-robotics/

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B.,

Kafai, Y. (2009). Scratch: programming for all. Communications of the ACM, November 2009.

Saeli, M. et al., 2011. Teaching Programming in Secondary School: A Pedagogical Content Knowledge Perspective. Informatics in Education, 10(1), pp.73–88.

Syslo, M.M. Kwiatkowska, A.B. (2006). Contribution of informatics education to mathematics education in schools. In: Mittermeir, R.T. (Ed.), ISSEP 2006, LNCS, 4226, 209–219.

Shulman, L.S. (1986). Those who understand: knowledge growth in teaching. Educational Researcher, 15, 4–14.

udacity.com (2021) Should I Learn C or C++? Available at: https://www.udacity.com/blog/2021/05/should-i-learn-c-or-cpp.html

Vaish, D., (2018). Python Robotics Projects: Build smart and collaborative robots using Python. Packt Publishing Ltd

Van Schooneveld, J., (2020). Build Physical Projects With Python on the Raspberry Pi Available at: https://realpython.com/python-raspberry-pi/