Bachelor of Science

Computer Science

Bachelor of Science – Computer Science

MISSION STATEMENT

The Bachelor of Science in Computer Science program at Springfield Research University aims to cultivate skilled problem solvers, algorithm developers, and effective communicators. Our mission is to equip students with the theoretical foundations and practical skills necessary for analyzing complex problems, designing innovative solutions, and contributing to technological advancements. Through tailored studies in areas such as networking, security, software development, and data science, we prepare graduates to excel in a dynamic and ever-evolving field.

At Springfield Research University (SRU), our Bachelor of Science in Computer Science program is designed to cultivate technical excellence, critical thinking, and innovation. Here are the key pillars of our mission:

1. **Academic Excellence and Problem-Solving Skills**:
    - Upholding rigorous standards, we foster critical thinking and intellectual growth.
    - Through engaging coursework, hands-on projects, and practical training, we empower computer science students to excel in the dynamic technological landscape.
2. **Cutting-Edge Research and Innovation**:
    - Our faculty and students actively contribute to computer science research and technological advancements.
    - By addressing complex problems and shaping the future of computing, we drive positive change within the field.
3. **Societal Impact and Ethical Responsibility**:
    - SRU recognizes its role in society.
    - Our graduates are not only competent computer science professionals but also ethical leaders who advocate for responsible technology.
    - They make meaningful contributions to their communities, ensuring fairness, security, and equality in the digital age.

At Springfield Research University, our Bachelor of Science in Computer Science program equips students with essential knowledge and skills in the dynamic field of computing. Here are some key areas covered in our curriculum:

1. **Introduction to Programming and Algorithms**:
    - Students learn fundamental programming concepts, data structures, and algorithm design.
    - They delve into problem-solving techniques and computational thinking.
2. **Object-Oriented Design and Software Development**:
    - Understanding software architecture and design principles is crucial.
    - Students explore object-oriented programming, software engineering, and application development.
3. **Database Systems and Data Management**:
    - Database design, SQL, and data modeling are essential components.
    - Students gain practical experience in creating and managing databases.
4. **Computer Networks and Security**:
    - Networking protocols, network administration, and cybersecurity are covered.
    - Students learn about securing systems and protecting data.

5. **Operating Systems and System Programming**:
    - Understanding operating system concepts and system-level programming is vital.
    - Students explore memory management, file systems, and process synchronization.
6. **Artificial Intelligence and Machine Learning**:
    - Students delve into AI algorithms, neural networks, and natural language processing.
    - They explore applications in areas like robotics and data analytics.
7. **Web Development and User Interfaces**:
    - Web technologies, front-end frameworks, and responsive design are emphasized.
    - Students create interactive web applications.
8. **Software Project Management and Ethics**:
    - Project planning, teamwork, and ethical considerations are part of the curriculum.
    - Students learn to manage software projects effectively.

At Springfield Research University, our Bachelor of Science in Computer Science program combines theoretical knowledge with hands-on practical experience. Here are some key components of our multifaceted approach:

1. **Software Development Projects**:
    - Students participate in software development projects, working on real-world applications.
    - They gain hands-on experience in coding, debugging, and collaborating within a team.
2. **Internships and Industry Collaborations**:
    - Our program collaborates with tech companies, startups, and research institutions.
    - Students engage in internships, apply theoretical concepts, and build professional networks.
3. **Hackathons and Coding Competitions**:
    - Students participate in hackathons, coding challenges, and competitions.
    - They solve complex problems, develop algorithms, and enhance their coding skills.
4. **Capstone Projects and Research**:
    - In their final year, students work on capstone projects.
    - They apply their knowledge to create innovative solutions or conduct research in specialized areas.
5. **Guest Lectures and Workshops**:
    - Industry experts deliver guest lectures on emerging technologies and trends.
    - Students learn about current challenges, practical applications, and career pathways.

RATIONALE FOR THE BACHELOR OF SCIENCE IN COMPUTER SCIENCE

The Bachelor of Science in Computer Science program at Springfield Research University is purposefully designed to meet the demands of the rapidly evolving technology landscape. Rooted in academic excellence, it equips students with essential computational knowledge and ethical foundations. By emphasizing cutting-edge research and societal impact, our

graduates become competent computer science professionals who contribute meaningfully to the field and beyond.

1. **National and Regional Relevance**:
    - **National Needs (Eswatini)**:
        - **Legal Capacity Building**: Just as Eswatini requires a robust legal workforce, the tech industry demands skilled computer scientists. Our program aims to equip students with the necessary technical knowledge, problem-solving skills, and ethical grounding to contribute effectively.
        - **Access to Technological Solutions**: By producing competent computer science professionals, we enhance access to innovative solutions. Graduates can create software, optimize systems, and drive technological progress.
        - **Policy and Innovation**: The program fosters critical thinking, enabling graduates to engage in research, policy formulation, and governance within the tech sector.
    - **Regional Needs (SADC)**:
        - **Harmonization of Digital Systems**: SADC member states face common technological challenges. Our program aligns with SADC's goal of harmonizing digital frameworks, facilitating regional collaboration, and ensuring seamless cross-border interactions.
        - **Human Capital Development**: Computer science professionals play a pivotal role in regional integration. Our graduates contribute to building a skilled workforce that can navigate complex digital landscapes.
        - **Digital Justice and Security**: SADC's stability relies on secure digital environments. Our graduates contribute to maintaining cybersecurity, resolving tech-related disputes, and promoting regional digital resilience.
2. **Purpose of the Program**:
    - **Educating Ethical Innovators**: The program instills ethical values, emphasizing integrity, fairness, and responsible technology use. Graduates become not only tech practitioners but also advocates for ethical digital solutions.
    - **Research and Technological Advancements**: Students engage in research, addressing contemporary tech challenges. Their findings contribute to technological innovation, policy formulation, and practical solutions.
    - **Community Impact**: The program prepares graduates to impact communities positively. Whether through software development, tech advocacy, or public service, they drive meaningful digital change.

PROGRAMME LEARNING OBJECTIVES

The **Program Learning Objectives** for the **Bachelor of Science in Computer Science** to make them more comprehensive and specific:

1. **Technical Proficiency and Problem-Solving**:

- Graduates will demonstrate proficiency in core computer science concepts, algorithms, and programming languages.
- They will apply analytical skills to solve complex computational problems, design efficient algorithms, and optimize software solutions.

2. **Software Development and Engineering Skills**:
   - Graduates will possess practical software development skills, including software architecture, version control, and debugging.
   - They will create robust, maintainable, and scalable software systems using best practices.
3. **Data Structures and Algorithms Mastery**:
   - Graduates will understand fundamental data structures (arrays, linked lists, trees, graphs, etc.) and their applications.
   - They will analyze algorithmic complexity, implement efficient algorithms, and evaluate trade-offs.
4. **Systems and Networking Knowledge**:
   - Graduates will comprehend operating systems, computer networks, and distributed computing.
   - They will configure networks, manage system resources, and troubleshoot network issues.
5. **Database Design and Management**:
   - Graduates will design and implement relational databases.
   - They will understand SQL, normalization, indexing, and data modeling.
6. **Human-Computer Interaction (HCI)**:
   - Graduates will appreciate HCI principles, usability, and user-centered design.
   - They will create intuitive interfaces and consider user experience.
7. **Security Awareness and Ethical Responsibility**:
   - Graduates will recognize security threats, encryption techniques, and privacy concerns.
   - They will adhere to ethical guidelines and protect user data.
8. **Collaboration and Communication Skills**:
   - Graduates will work effectively in interdisciplinary teams.
   - They will communicate technical concepts clearly to diverse audiences.

These comprehensive objectives prepare students for a successful career in computer science, equipping them with both theoretical knowledge and practical skills.

PROGRAMME LEARNING OUTCOMES

The **Program Learning Outcomes** (PLOs) for the **Bachelor of Science in Computer Science** are:

1. **Algorithmic Proficiency**:
    - Graduates can analyze problems, design efficient algorithms, and implement solutions in various domains.
    - They demonstrate mastery in data structures, sorting, searching, and graph algorithms.
2. **Software Development Skills**:
    - Graduates can develop software applications using appropriate programming languages and tools.
    - They understand software engineering principles, version control, and debugging techniques.
3. **Systems Understanding**:
    - Graduates comprehend operating systems, memory management, and file systems.
    - They can configure networks, manage system resources, and troubleshoot system issues.
4. **Database Management**:
    - Graduates design and implement relational databases.
    - They can create, query, and optimize databases using SQL.
5. **User-Centered Design**:
    - Graduates appreciate HCI principles and create user-friendly interfaces.
    - They consider usability, accessibility, and user experience.
6. **Security Awareness and Ethics**:
    - Graduates recognize security threats, encryption methods, and privacy concerns.
    - They adhere to ethical guidelines in software development.
7. **Collaboration and Communication**:
    - Graduates work effectively in teams, communicate technical concepts, and collaborate on projects.
    - They can present ideas clearly to diverse audiences.
8. **Continuous Learning and Adaptability**:
    - Graduates stay updated with technological advancements.
    - They adapt to evolving tools, languages, and industry trends.

These outcomes ensure that students are well-prepared for a successful career in computer science, equipped with both theoretical knowledge and practical skills.

BSC COMPUTER SCIENCE ENTRY REQUIREMENTS

The student must have 6 passes in SGCSE/GCE/IGCSE O' level including a pass with Grade C or better in English Language and at least four other subjects. Special: Mathematics and any other two from Biology, Chemistry, Combined Science, Physics, Physical Science and/or Human and Social Biology. Faculty may set mature entry requirements subject to approval by Senate.

CAREER OPPORTUNITIES

Here are some **Career Opportunities** for graduates with a **Bachelor of Science in Computer Science**:

1. **Software Developer/Engineer**:
    - Design, develop, and maintain software applications, systems, and websites.
    - Work in various domains such as web development, mobile apps, or enterprise software.
2. **Data Scientist/Analyst**:
    - Analyze and interpret data to extract insights and drive decision-making.
    - Use machine learning and statistical techniques to solve complex problems.
3. **Cybersecurity Specialist**:
    - Protect organizations from cyber threats by implementing security measures.
    - Conduct vulnerability assessments, monitor networks, and respond to incidents.
4. **Cloud Solutions Architect**:
    - Design and deploy cloud-based solutions using platforms like AWS, Azure, or Google Cloud.
    - Optimize scalability, reliability, and performance.
5. **Front-End/Back-End Developer**:
    - Front-End: Create user interfaces, enhance user experience, and optimize website performance.
    - Back-End: Build server-side logic, APIs, and databases.
6. **Mobile App Developer**:
    - Develop applications for iOS or Android platforms.
    - Use languages like Swift, Kotlin, or React Native.
7. **Database Administrator (DBA)**:
    - Manage databases, ensure data integrity, and optimize query performance.
    - Work with SQL, NoSQL, and big data technologies.
8. **Machine Learning Engineer**:
    - Build and deploy machine learning models for predictive analytics.
    - Collaborate with data scientists and domain experts.
9. **Game Developer**:
    - Create video games, simulations, or interactive experiences.
    - Combine programming, graphics, and game design.
10. **IT Consultant/Systems Analyst**:
    - Advise organizations on technology solutions, process improvements, and system implementations.
    - Analyze business requirements and recommend IT strategies.

Remember that the field of computer science offers diverse career paths, and graduates can

specialize based on their interests and skills.

The Bachelor's Degree shall:

The Bachelor's degree program in Bachelor of Science in Computer Science at Springfield Research University is designed to equip students with the skills and knowledge necessary for a successful career in this dynamic field. Here are the key features of our program:

1. **Duration**:
    - The program spans **four years** for full-time students or **six years** for part-time students, including an industrial attachment or internship period.
2. **Semester Structure**:
    - Each academic year consists of **two semesters**.
    - **Semester Duration**: Each semester runs for **20 weeks**.
        - **Orientation Week**: One week dedicated to orientation.
        - **Teaching Weeks**: A minimum of **14 weeks** for instruction.
        - **Mid-Semester Break**: A one-week break for students.
        - **Examination Period**: Two weeks for final exams.
        - **Results Processing**: Two weeks allocated for marking and result processing.

Our program ensures a rigorous academic experience while allowing flexibility for part-time students. Students engage in hands-on learning, theoretical coursework, and practical projects, preparing them for the exciting challenges of the Computer Science industry.

Special Departmental Regulations

1. **Course Completion Requirements**:
    - All **Core**, **Prerequisite**, **Required**, **General**, and **Elective** courses within the degree program are compulsory. Students must pass these courses with a minimum grade of **50%** to graduate.
    - However, during the fourth and fifth years, all courses must be passed with a minimum grade of **60%** (equivalent to a CGPA of **3.00**) to qualify for graduation.
2. **Optional Courses**:
    - Optional courses do not contribute to the final grade. Their marks are excluded from the computation of the overall grade.
3. **Externalization of Courses**:
    - All courses within the degree programs must be completed internally. Externalization is not permitted.
4. **Quality Control and Evaluation**:

- Regular academic audits and reviews occur every four years, overseen by external moderators. Internal program evaluation is ongoing.

5. **Competence and Preparation**:
    - The courses offered in the Bachelor of Science in Computer Science program provide adequate competences, preparing students for professional practice at the required academic level.
6. **Core and Prerequisite Courses**:
    - Students must pass all Core and Prerequisite courses with a minimum grade of **50%** before progressing to the next level or enrolling in additional courses.

Degree Award and Classification

- Upon successful completion of all **Core**, **Required**, and **Education** courses, as well as meeting the program requirements, a student will be awarded the degree of **Bachelor of Science in Computer Science at** the end of the final year.
- The **normal classification** of a bachelor's degree is determined based on the academic performance during the third and fourth years of study.

Rationale to Course Numbering

At Springfield Research University, we meticulously design our Bachelor of Science in Computer Science curriculum to empower students with the knowledge and skills needed to thrive in this dynamic field. Our course numbering system serves as a roadmap, guiding students through their academic journey - ****100-level courses**** introduce foundational concepts. - ****200-level courses**** build on those foundations. - ****300-level courses**** explore more specialized topics. - ****400-level courses**** are advanced and often include research or project components. Let's delve into the reasons behind our thoughtful approach:

1. **Logical Progression**: Our course numbers reflect a logical progression. Foundational concepts begin with the "100" series, followed by deeper explorations in the "200" and "300" levels. Advanced topics and research opportunities reside in the "400" series.
2. **Prerequisites and Coherence**: Clear numbering helps students understand prerequisites and co-requisites. For instance, a 200-level course assumes knowledge from related 100-level courses, ensuring a coherent learning experience.
3. **Specialization and Depth**: As students advance, higher-level courses delve into specialized areas such as machine learning, cybersecurity and visualization. The numbering system communicates this depth of study.
4. **Alignment with Program Goals**: Each course number aligns with our program's learning outcomes. Whether it's mastering artificial intelligence or diving into java programming, students can track their progress.
5. **Transferability**: Consistent numbering facilitates credit transfer between institutions, supporting seamless academic mobility.

In summary, our course numbering isn't just a sequence—it's a deliberate framework that enhances learning, fosters curiosity, and prepares our students for impactful careers in

Computer Science industry. Bachelor of Science in Computer Science courses simplify the course numbering system.

100-Level Courses:

- **CS 101:** Introduction to Computer Science
- **CS 110:** Programming Fundamentals
- **CS 120:** Discrete Mathematics for CS

200-Level Courses:

- **CS 201:** Data Structures and Algorithms
- **CS 210:** Computer Organization and Architecture
- **CS 220:** Software Engineering Principles

300-Level Courses:

- **CS 301:** Operating Systems and System Programming
- **CS 310:** Database Management Systems
- **CS 320:** Web Development and Design

400-Level Courses:

- **CS 401:** Artificial Intelligence and Machine Learning
- **CS 410:** Computer Networks and Security
- **CS 420:** Capstone Project in Computer Science

CREDIT TRANSFER, ACCUMULATION AND DISTRIBUTION OF NOTIONAL HOURS

The Bachelor of Science in Computer Science is a four (4) year program. The student is expected to accumulate 495 credit points to be considered to have met the requirements of the Bachelor of Science in Computer Science must pass each module by at least 50%.

- Level 1 = minimum of credits 124 (1240 notional hours of study)
- Level 2 = minimum of credits 124 (1240 notional hours of study)
- Level 3 = minimum of credits 124 (1240 notional hours of study)
- Level 4 = minimum of credits 124 (1240 notional hours of study)

TOTAL credit points 496 (4960 notional hours of study)

Credit Transfer and Accumulation

1. Credits are derived from engagement of students in learning activities during lectures, seminars, tutorials, micro or macro field trips, directed and self-directed learning and writing examination tests and assignments.
2. Modules from the health and medical faculty are worth 12 credit. Lecture attendance is compulsory. Students who attend less than 80% of lessons will not be allowed to sit for their sessional examinations.

Weighting

The degree class shall be based on weighting the results from part 1, 2, 3 and 4, the Degree weighting shall be as follows:

Level 1	20%
Level 2	20%
Level 3	30%
Level 4	30%

Distribution of Notional Hours

Module	Lecture Hrs	Tutorials/ Seminars	Self-Directed Study	Assignment Tests/Exams	Notional Hrs	Credits
CS000	50	20	25	25	120	12
PROJECT	0	0	40	100	140	14

Teaching Methods

At Springfield Research University (SRU), we are committed to employing a diverse array of teaching methods to ensure a comprehensive and engaging learning experience for our students. Our teaching methods are carefully selected to align with the programme's objectives and to meet the needs of our diverse student body. The following are the key teaching methods utilized across all SRU programmes:

1. Lectures:

- Lectures are used to introduce and explain key concepts, theories, and principles. They provide a structured and systematic approach to delivering content, allowing students to gain a solid foundation in their respective fields. Lectures are often supplemented with visual aids, multimedia presentations, and interactive elements to enhance understanding and engagement.

2. Seminars:

- Seminars are interactive sessions that promote critical thinking and in-depth discussion on specific topics. Students are encouraged to actively participate, share their perspectives, and engage in debates. Seminars provide an opportunity for students to develop their analytical and communication skills.

3. Workshops:

- Workshops are hands-on sessions that focus on practical skills and applications. These sessions allow students to engage in experiential learning, apply theoretical knowledge to real-world scenarios, and collaborate with peers on projects and activities. Workshops are designed to foster creativity, problem-solving, and teamwork.

4. Problem-Based Learning (PBL):

- Problem-Based Learning is a student-centered approach that involves presenting students with complex, real-world problems to solve. Students work in small groups to research, discuss, and propose solutions, developing critical thinking and collaborative skills in the process. PBL encourages independent learning and active engagement.

5. Case Studies:

- Case studies are used to analyze real-life situations and decision-making processes. Students examine and discuss case studies to understand the context, identify key issues, and evaluate possible solutions. This method helps students develop their analytical and problem-solving abilities while relating theoretical concepts to practical situations.

6. Clinical Practice:

- For programmes that include a clinical component, such as Health and Medical Sciences, clinical practice is an integral part of the curriculum. Students gain hands-on experience in clinical settings, working under the supervision of qualified professionals. This method provides valuable opportunities for students to apply their knowledge, develop clinical skills, and gain insights into professional practice.

7. Research Projects:

- Research projects are designed to cultivate a culture of inquiry and innovation. Students engage in independent or group research projects, exploring topics of interest and contributing to the body of knowledge in their field. Research projects develop students' research skills, critical thinking, and ability to communicate findings effectively.

8. Online Learning:

- Online learning is incorporated to provide flexible and accessible education. SRU utilizes online platforms to deliver lectures, conduct discussions, and facilitate collaborative projects. Online learning allows students to access course materials, participate in virtual classrooms, and engage with peers and instructors remotely.

9. Continuous Assessment:

- Continuous assessment methods, such as quizzes, assignments, and presentations, are used to monitor students' progress and provide ongoing feedback. These assessments help identify areas for improvement and ensure that students are meeting learning objectives throughout the course.

10. Peer Learning:

- Peer learning encourages students to collaborate and learn from each other. Group projects, study groups, and peer review sessions provide opportunities for students to share knowledge, offer feedback, and support each other's learning journey.

At SRU, our commitment to employing diverse and effective teaching methods ensures that our students receive a well-rounded education that prepares them for success in their chosen fields. We continuously review and enhance our teaching practices to provide the best possible learning experience for our students.

Delivery Methods

At Springfield Research University (SRU), we utilize a variety of delivery methods to ensure that our educational programmes are accessible, engaging, and effective. Our delivery methods are designed to cater to the diverse needs of our students and to provide flexible

learning opportunities. The following are the key delivery methods employed across all SRU programmes:

1. In-Person Delivery:

- **Classroom Lectures:** Traditional classroom lectures provide a structured and interactive environment where students can engage with instructors and peers. These sessions often include discussions, presentations, and multimedia resources to enhance learning.
- **Laboratory Sessions:** For programmes that require practical and experimental learning, laboratory sessions are conducted in specialized labs equipped with the necessary tools and equipment. These hands-on sessions allow students to apply theoretical knowledge in a controlled environment.
- **Clinical Placements:** Health and Medical Sciences programmes include clinical placements in hospitals, clinics, and healthcare facilities. These placements provide students with real-world experience under the supervision of qualified professionals.

2. Online Delivery:

- **Virtual Classrooms:** Online platforms are used to deliver lectures, conduct discussions, and facilitate collaborative projects. Virtual classrooms enable students to access course materials, participate in live sessions, and engage with peers and instructors from remote locations.
- **Recorded Lectures:** Recorded lectures are made available for students to access at their convenience. This flexible approach allows students to review and revisit course content as needed.
- **Online Assessments:** Online assessments, including quizzes, assignments, and exams, are conducted through secure online platforms. These assessments provide timely feedback and help monitor students' progress.

3. Blended Learning:

- **Hybrid Courses:** Blended learning combines in-person and online delivery methods to provide a flexible and comprehensive learning experience. Hybrid courses may involve alternating between classroom sessions and online activities.
- **Flipped Classroom:** In the flipped classroom model, students access instructional content online before class and use in-person sessions for interactive, application-based activities. This approach encourages active learning and deeper engagement with the material.

4. Independent Study:

- **Self-Paced Learning:** Self-paced learning allows students to progress through course materials at their own speed. This method is ideal for students who prefer to learn independently and manage their own schedules.
- **Research Projects:** Independent research projects provide students with the opportunity to explore topics of interest, develop research skills, and contribute to the body of knowledge in their field. Faculty advisors provide guidance and support throughout the research process.

5. Collaborative Learning:

- **Group Projects:** Group projects foster teamwork and collaboration among students. These projects often involve problem-solving, research, and presentations, allowing students to learn from each other and develop interpersonal skills.
- **Peer Review:** Peer review sessions encourage students to provide and receive constructive feedback on each other's work. This method promotes critical thinking, reflection, and improvement.

6. Experiential Learning:

- **Internships and Work Placements:** Internships and work placements provide students with practical experience in their chosen field. These opportunities allow students to apply their knowledge in real-world settings, develop professional skills, and build industry connections.
- **Field Trips and Excursions:** Field trips and excursions offer experiential learning opportunities outside the classroom. These activities provide students with firsthand exposure to relevant sites, industries, and practices.

7. Continuous Assessment:

- **Formative Assessments:** Formative assessments, such as quizzes, assignments, and in-class activities, provide ongoing feedback to students and help track their progress. These assessments are designed to support learning and identify areas for improvement.
- **Summative Assessments:** Summative assessments, including final exams, projects, and presentations, evaluate students' overall performance and mastery of course content.

At SRU, our diverse delivery methods ensure that students receive a well-rounded and flexible education that caters to their individual learning preferences. We are committed to continuously enhancing our delivery methods to provide the best possible learning experience for our students.

COURSE MODULES AND SYNOPSIS

YEAR 1: SEMESTER 1

Code	Course	Lectures (Hrs)	Tutorials (Hrs)	Credits
CS101	Computing Fundamentals	**96**	**0**	**9.6**
CS102	Introduction to Program Design	**90**	**6**	**9.6**
CS103	Introduction to Machine Organisation and Assembly Language	**96**	**0**	**9.6**
CS104	Discrete Mathematics	**96**	**0**	**9.6**
CS105	Object-Oriented Design Concepts	**96**	**0**	**9.6**
CS106	Computer Programming I	**40**	**56**	**9.6**
	TOTAL			**57.6**

YEAR 1: SEMESTER 2

Code	Course	Lectures (Hrs)	Tutorials (Hrs)	Credits

CS107	Accelerated Program Design	**96**	**0**	**9.6**
CS108	Accelerated Object-Oriented Design Concepts	**96**	**0**	**9.6**
CS109	Application Building with Object Oriented Concepts	**40**	**56**	**9.6**
CS110	Algorithms and Data Structures	**96**	**0**	**9.6**
CS111	Systems Programming Concepts	**96**	**0**	**9.6**
CS112	Computer Programming II	**40**	**56**	**9.6**
	TOTAL			**57.6**

YEAR 2: SEMESTER 3

Code	Course	Lectures (Hrs)	Tutorials (Hrs)	Credits
CS213	Human Computer Interaction	**96**	**0**	**9.6**
CS214	Social Implications for Information Processing	**90**	**6**	**9.6**
CS215	Foundations of Computer Science	**96**	**0**	**9.6**
CS216	Database Systems I	**96**	**0**	**9.6**
CS217	Computer Networks	**96**	**0**	**9.6**
CS218	Software Engineering	**40**	**56**	**9.6**
	TOTAL			**57.6**

YEAR 2: SEMESTER 4

Code	Course	Lectures (Hrs)	Tutorials (Hrs)	Credits
CS219	Theory of Automata	**96**	**0**	**9.6**
CS220	Numerical Methods for Linear and Non-Linear Systems	**96**	**0**	**9.6**
CS221	Numerical Methods for Calculus and Differential Equations	**40**	**56**	**9.6**
CS222	Artificial Intelligence for Interactive Media and Games	**96**	**0**	**9.6**
CS223	Biovisualization	**96**	**0**	**9.6**
CS224	Application Development and Emerging Technologies I	**40**	**56**	**9.6**
	TOTAL			**57.6**

YEAR 3: SEMESTER 5 & 6

Code	Course	Lectures (Hrs)	Tutorials (Hrs)	Credits
CS225	Industrial Attachment		**450**	**45**
CS226	Industrial Attachment Report	**90**	**90**	**9**

YEAR 4: SEMESTER 7

Code	Course	Lectures (Hrs)	Tutorials (Hrs)	Credits
CS227	Web ware: Computational Technology for Network Information Systems	**96**	**0**	**9.6**
CS228	Tools and Techniques in Computer Networks	**90**	**6**	**9.6**
CS229	Database Systems II	**96**	**0**	**9.6**

CS230	Data Mining and Knowledge Discovery in Databases	**96**	**0**	**9.6**
CS231	Emerging Technology	**96**	**0**	**9.6**
CS232	Software Security Engineering	**40**	**56**	**9.6**
	TOTAL			**57.6**

YEAR 4: SEMESTER 8

Code	Course	Lectures (Hrs)	Tutorials (Hrs)	Credits
CS233	Distributed Computer Systems	**96**	**0**	**9.6**
CS234	Mobile & Ubiquitous Computing	**96**	**0**	**9.6**
CS235	Graphics Computing and Multimedia Technology	**96**	**0**	**9.6**
CS236	Technoprenuership	**96**	**0**	**9.6**
CS237	Research Project	**80**	**100**	**18**
CS232	Software Security Engineering	**40**	**56**	**9.6**
	TOTAL			**56.4**

COURSE DESCRIPTIONS

ACCELERATED INTRODUCTION TO PROGRAM DESIGN.

In the first half of the term, this course covers the same functional programming material as CS 1101 at roughly twice the pace. The second half of the term is a preview of selected advanced Computer Science topics, such as the design and implementation of application-specific languages, macros, programming with the HTTP protocol, and continuation-passing style. Students will be expected to complete an open-ended individual programming project.

INTRODUCTION TO MACHINE ORGANIZATION AND ASSEMBLY LANGUAGE.

This course introduces students to the structure and behavior of modern digital computers and the way they execute programs. Machine organization topics include the Von Neumann model of execution, functional organization of computer hardware, the memory hierarchy, caching performance, and pipelining. Assembly language topics include representations of numbers in computers, basic instruction sets, addressing modes, stacks and procedures, low-level I/O, and the functions of compilers, assemblers, linkers, and loaders. The course also presents how code and data structures of higher-level languages are mapped into the assembly language and machine representations of a modern processor. Programming projects will be carried out in the C language and the assembly language of a modern processor.

DISCRETE MATHEMATICS.

This course serves as an introduction to some of the more important concepts, techniques, and structures of discrete mathematics providing a bridge between computer science and mathematics. Topics include sets, functions and relations, propositional and predicate calculus, mathematical induction, properties of integers, counting techniques, and graph theory. Students will be expected to develop simple proofs for problems drawn primarily from computer science and applied mathematics.

OBJECT-ORIENTED DESIGN CONCEPTS.

This course introduces students to an object-oriented model of programming. Building from the design methodology covered in CS 1101/CS 1102, this course shows how programs can be decomposed into classes and objects. By emphasizing design, this course shows how to implement small defect-free programs and evaluate design decisions to select an optimal design under specific assumptions. Topics include inheritance, exceptions, interface, design by contract, basic design patterns, and reuse. Students will be expected to design, implement, and debug object-oriented programs composed of multiple classes and over a variety of data structures.

ACCELERATED OBJECT-ORIENTED DESIGN CONCEPTS.

This course covers the data structures and general program-design material from CS2102, but assumes that students have significant prior experience in object-oriented programming. The course covers object-oriented design principles and data structures more deeply and at a faster pace than in CS 2102. Students will be expected to design, implement, test, debug, and critique programs both for correctness and adherence to good object-oriented design principles. The course is designed to strengthen both the design skills and algorithmic thinking of students who already have a foundation in object-orient- ed programming.

APPLICATION BUILDING WITH OBJECT-ORIENTED CONCEPTS.

This course introduces students to an object-oriented model of programming, with an emphasis on the programming approaches useful in creating software applications. Students will be expected to design, implement, and debug object-oriented programs. Topics include inheritance, user interfaces, and database access. This course is for non-CS majors with prior programming experience and an interest in building software applications.

ALGORITHMS AND DATA STRUCTURES

Building on a fundamental knowledge of data structures, data abstraction techniques, and mathematical tools, a number of examples of algorithm design and analysis — worst case and average case — will be developed. Topics include greedy algorithms, divide-and-conquer, dynamic programming, heuristics, and probabilistic algorithms. Problems will be drawn from areas such as sorting, graph theory, and string processing. The influence of the computational model on algorithm design will be discussed.

SYSTEMS PROGRAMMING FOR NON-MAJORS.

This course introduces the C programming language and system programming concepts to non-CS majors who need to program computers in their own fields. The course assumes that students have had previous programming experience. It quickly introduces the major concepts of the C language and covers manual memory management, pointers and basic data structures, the machine stack, and input/output mechanisms. Students will be expected to design, implement, and debug programs in C.

SYSTEMS PROGRAMMING CONCEPTS.

This course introduces students to a model of programming where the programming language exposes details of how the hardware stores and executes software. Building from the design concepts covered in CS 2102, this course covers manual memory management, pointers, the machine stack, and input/ output mechanisms. The course will involve large-scale programming exercises and will be designed to help students confront issues of safe programming with system-level constructs. The course will cover several tools that assist programmers in these tasks. Students will be expected to design, implement, and debug programs in C++ and C. The course presents the material from CS 2301 at a fast pace and

also includes C++ and other advanced topics.

HUMAN-COMPUTER INTERACTION.

This course develops in the student an understanding of the nature and importance of problems concerning the efficiency and effectiveness of human interaction with computer-based systems. Topics include the design and evaluation of interactive computer systems, basic psychological considerations of interaction, interactive language design, interactive hardware design, and special input/output techniques. Students will be expected to complete several projects. A project might be a software evaluation, interface development, or an experiment.

SOCIAL IMPLICATIONS OF INFORMATION PROCESSING.

This course makes the student aware of the social, moral, ethical, and philosophical impact of computers and computer-based systems on society, both now and in the future. Topics include major computer-based applications and their impact, human- machine relationships, and the major problems of controlling the use of computers. Students will be expected to contribute to classroom discussions and to complete a number of significant writing assignments.

FOUNDATIONS OF COMPUTER SCIENCE.

This course introduces the theoretical foundations of computer science. These form the basis for a more complete understanding of the proficiency in computer science. Topics include computational models, formal languages, and an introduction to compatibility and complexity theory, including NP-completeness.

DATABASE SYSTEMS I.

This course introduces the student to the design, use, and application of database management systems. Topics include the relational data model, relational query languages, design theory, and conceptual data design and modeling for relational database design. Techniques that provide for data independence and minimal redundancy will be discussed.

COMPUTER NETWORKS.

This course provides a broad view of computer networks. The course exposes students to all seven layers of OSI Reference Model while providing an introduction into newer topics such as wireless networking and Internet traffic concerns. The objective is to focus on an understanding of fundamental concepts of modern computer network architecture from a design and performance perspective. Topics covered include physical layer considerations, network protocols, wide area networks, local area networks, wireless networks, switches and routing, congestion, Internet traffic, and network security. Students will be expected to do extensive systems/network programming and will be expected to make use of simulation and measurement tools to gain an appreciation of current network design and performance issues.

SOFTWARE ENGINEERING.

This course introduces the fundamental principles of software engineering. Modern software development techniques and life cycles are emphasized. Topics include requirements analysis and specification, analysis and design, architecture, implementation, testing and quality, configuration management, and project management. Students will be expected to complete a project that employs techniques from the topics studied. This course should be taken before any course requiring a large programming project.

NUMERICAL METHODS FOR LINEAR AND NONLINEAR SYSTEMS.

This course provides an introduction to modern computational methods for linear and nonlinear equations and systems and their applications. Topics covered include solution of nonlinear scalar equations, direct and iterative algorithms for the solution of systems of linear equations, solution of nonlinear systems, and the eigenvalue problem for matrices. Error analysis will be emphasized throughout.

NUMERICAL METHODS FOR CALCULUS AND DIFFERENTIAL EQUATIONS.

This course provides an introduction to modern computational methods for differential and integral calculus and differential equations. Topics covered include interpolation and polynomial approximation, approximation theory, numerical differentiation and integration, and numerical solutions of ordinary differential equations. Error analysis will be emphasized throughout.

ARTIFICIAL INTELLIGENCE FOR INTERACTIVE MEDIA AND GAMES.

Algorithms and programming techniques from artificial intelligence (AI) are key contributors to the experience of modern computer games and interactive media, either by directly controlling a non-player character (NPC) or through more subtle manipulation of the environment. This course will focus on the practical AI programming techniques currently used in computer games for NPC navigation and decision-making, along with the design issues that arise when AI is applied in computer games, such as believability and real-time performance. The course will also briefly discuss future directions in applying AI to games and media. Students will be expected to complete significant software development projects using the studied techniques.

ANALYSIS OF ALGORITHMS.

This course develops the skill of analyzing the behavior of algorithms. Topics include the analysis — with respect to average and worst-case behavior — and correctness of algorithms for internal sorting, pattern matching on strings, graph algorithms, and methods such as recursion elimination, dynamic programming, and program profiling. Students will be expected to write and analyze programs.

THEORY OF COMPUTATION.

Building on the theoretical foundations from CS 3133, this course addresses the fundamental question of what it means to be “computable,” including different characterization of computable sets and functions. Topics include the halting program, the Church-Turing thesis, primitive recursive functions, recursive sets, recursively enumerable sets, NP-completeness, and reducibilities. Students will be expected to complete a variety of exercises and proofs

OBJECT-ORIENTED ANALYSIS AND DESIGN.

This Software Engineering course will focus on the process of Object-Oriented Analysis and Design. Students will be expected to complete a large number of exercises in Domain Modeling, Use Case Analysis, and Object-Oriented Design. In addition, the course will investigate Design Patterns, which are elements of reusable object-oriented software designs. This course will survey a set of design patterns and consider how these patterns are described and used to solve design problems.

WEBWARE: COMPUTATIONAL TECHNOLOGY FOR NETWORK INFORMATION SYSTEMS.

This course explores the computational aspects of network information systems as embodied by the World Wide Web (WWW). Topics include languages for document design, programming languages for executable content, scripting languages, design of WWW based human/computer interfaces, client/server network architecture models, high level network protocols (e.g., http), WWW network resource discovery and network security issues. Students in this course will be expected to complete a substantial software project (e.g., Java based user interface, HTML/CGI based information system, WWW search mechanisms).

INTRODUCTION TO ARTIFICIAL INTELLIGENCE.

This course studies the problem of making computers act in ways which we call "intelligent". Topics include major theories, tools and applications of artificial intelligence; aspects of knowledge representation; searching and planning; and natural language understanding. Students will be expected to complete projects which express problems that require search in state spaces and to propose appropriate methods for solving the problems.

MACHINE LEARNING.

In this course, students will explore both theoretical and practical aspects of machine learning, including algorithms for regression, classification, dimensionality reduction, clustering, and density estimation. Specific topics may include neural networks and deep learning, Bayesian networks and probabilistic graphical models, principal component analysis, k-means clustering, decision trees and random forests, support vector machines, and kernel methods.

SOFTWARE SECURITY ENGINEERING.

This course provides an introduction to the pitfalls and practices of building secure software applications. Topics will include threat modeling, secure software development, defensive programming, web security, and the interaction between security and usability. The course focuses on the application level with minor attention to operating-system level security; network-level security is not covered. Assignments involve designing and implementing secure software, evaluating designs and systems for security-related flaws, and presentations on security issues or tools. All students will be required to sign a pledge of responsible conduct at the start of the course.

TOOLS AND TECHNIQUES IN COMPUTER NETWORK SECURITY.

This course introduces students to modern network security concepts, tools, and techniques. The course covers security threats, attacks, and mitigations at the operating-system and network levels (as opposed to the software level). Topics include authentication, authorization, confidentiality, integrity, anonymity, privacy, intrusion detection and response, and cryptographic applications. Students will become familiar with modern security protocols and tools. Assignments will involve using security-testing software to uncover vulnerabilities, network packet analyzers, and existing security applications to create secure network implementations. The course requires enough programming and systems background to understand attacks and use systems tools but does not involve significant programming projects. Assignments and projects will use a Linux base for implementation.

DATABASE SYSTEMS II.

This course concentrates on the study of the internals of database management systems. Topics include principles and theories of physical storage management, advanced query languages, query processing and optimization, index structures for relational databases, transaction processing, concurrency control, distributed databases, and database recovery, security, client server and transaction processing systems. Students may be expected to design and implement software components that make up modern database systems.

DATA MINING AND KNOWLEDGE DISCOVERY IN DATABASES.

This course provides an introduction to Knowledge Discovery in Databases (KDD) and Data Mining. KDD deals with data integration techniques and with the discovery, interpretation, and visualization of patterns in large collections of data. Topics covered in this course include data warehousing and mediation techniques; data mining methods such as rule-based learning, decision trees, association rules, and sequence mining; and data visualization. The work discussed originates in the fields of artificial intelligence, machine learning, statistical data analysis, data visualization, databases, and information retrieval. Several scientific and industrial applications of KDD will be studied.

DISTRIBUTED COMPUTING SYSTEMS.

This course extends the study of the design and implementation of operating systems begun in CS 3013 to distributed and advanced computer systems. Topics include principles and theories of resource allocation, file systems, protection schemes, and performance evaluation as they relate to distributed and advanced computer systems. Students may be expected to design and implement programs that emphasize the concepts of file systems and distributed computing systems using current tools and languages.

COMPUTER ARCHITECTURE.

This course explores the architectural design of modern computer systems in terms of instruction sets and the organization of processors, controllers, memories, devices, and communication links. Topics include an overview of computer architectures and system components, theoretical foundations, instruction-level and thread-level pipelining, multifunction pipelines, multi-core systems, caching and memory hierarchies, and multi-core and parallel computer organization. Students may be expected to design and implement programs that simulate significant components of modern computer architectures.

ADVANCED COMPUTER NETWORKS.

This course provides an in-depth look into computer networks. While repeating some of the areas from CS 3516, the goal is to go deeper into computer networks topics. This in-depth treatment in topics such as routing, congestion control, wireless layer protocols, and physical signaling considerations will require the use of basic queuing theory and probability to provide a more formal treatment of computer networks performance. Other topics covered include LAN and WLAN technologies, mobile wireless networks, sensor networks, optical networks, network security, intrusion detection, and network management. Students will be expected to do more sophisticated network programming than seen in CS 3516 and will conduct laboratory activities involving measuring the performance of modern networking applications running on both wired networks and infrastructure wireless networks. The course assumes a familiarity with operating systems including Unix or Linux and significant experience with C/C++.

MOBILE AND UBIQUITOUS COMPUTING.

The goal of this course is to acquaint students with fundamental concepts and state-of-the-art computer science literature in mobile and ubiquitous computing. Topics to be covered include mobile systems issues, human activity and emotion sensing, location sensing, mobile human-computer interaction, mobile social networking, mobile health, power saving techniques, energy and mobile performance measurement studies, and mobile security. The course will introduce the programming of mobile devices such as smartphones running the Android operating system.

TECHNIQUES OF PROGRAMMING LANGUAGE TRANSLATION

This course studies the compiling process for high-level languages. Topics include lexical analysis, syntax analysis, semantic analysis, symbol tables, intermediate languages, optimization, code generation, and run-time systems. Students will be expected to use compiler tools to implement the front end — and to write a program to implement the back end — of a compiler for a recursive programming language.

PROGRAMMING LANGUAGES

This course covers the design and implementation of programming languages. Topics include data structures for representing programming languages, implementing control structures (such as functions, recursion, and exceptions), garbage collection, and type systems. Students will be expected to implement several small languages using a functional programming language.

COMPUTER GRAPHICS

This course studies the use of the computer to model and graphically render two- and three-dimensional structures. Topics include graphics devices and languages, 2- and 3-D object representations, and various aspects of rendering realistic images. Students will be expected to implement programs which span all stages of the 3-D graphics pipeline, including clipping, projection, arbitrary viewing, hidden surface removal, and shading.

COMPUTER ANIMATION

This course provides an in-depth examination of the algorithms, data structures, and techniques used in modeling and rendering dynamic scenes. Topics include animation hardware and software; parametric blending techniques; modeling physical and articulated objects; forward and inverse kinematics; key-frame, procedural, and behavioral animation; and free-form deformation. Students will be expected to develop programs to implement low-level animation algorithms as well as use commercial animation tools to design and produce small to moderate sized animations.

INTRODUCTION TO CRYPTOGRAPHY AND COMMUNICATION SECURITY.

This course provides an introduction to modern cryptography and communication security. It focuses on how cryptographic algorithms and protocols work and *how* to use them. The course covers the concepts of block ciphers and message authentication codes, public key encryption, digital signatures, and key establishment, as well as common examples and uses of such schemes, including the AES, RSA-OAEP, and the Digital Signature Algorithm. Basic cryptanalytic techniques and examples of practical security solutions are explored to understand how to design and evaluate modern security solutions. The course is suited for students interested in cryptography or other security related fields such as trusted computing, network and OS security, or general IT security.

BIOVISUALIZATION
This course will use interactive visualization to model and analyze biological information, structures, and processes. Topics will include the fundamental principles, concepts, and techniques of visualization (both scientific and information visualization) and how visualization can be used to study bioinformatics data at the genomic, cellular, molecular, organism, and population levels. Students will be expected to write small to moderate programs to experiment with different visual mappings and data types.

COURSE OUTLINES

Course Title: Computer Science I

Course Description:

Computer Science I introduce fundamental concepts in programming and problem-solving. Students gain proficiency in a high-level programming language and learn to design, implement, and analyze algorithms. Topics include data types, control structures, functions, and basic data structures.

Learning Objectives:

By the end of this course, students will be able to:

1. Understand the principles of programming languages.
2. Write, debug, and optimize code in a chosen programming language.
3. Analyze and solve computational problems using algorithms.
4. Apply software engineering practices, including version control and documentation.

Topics Covered:

1. **Introduction to Programming**:
    - Overview of programming paradigms.
    - Syntax and semantics of a programming language.
    - Variables, data types, and expressions.
2. **Control Structures**:
    - Conditional statements (if, else, switch).
    - Loops (while, for).
    - Error handling.
3. **Functions and Modular Programming**:
    - Function definition and invocation.
    - Parameters and return values.
    - Scope and lifetime of variables.
4. **Data Structures**:
    - Arrays and lists.
    - Strings.
    - Dictionaries and sets.
5. **Algorithm Design and Analysis**:
    - Basic algorithmic techniques (brute force, divide and conquer).
    - Time and space complexity analysis.

Assessment:

- **Assignments**: Weekly programming assignments to reinforce concepts.
- **Midterm Exam**: Covers material up to the midpoint of the course.
- **Final Project**: Implement a larger-scale program.
- **Participation**: Active engagement in class discussions and labs.

Recommended Reading:

1. “Introduction to the Theory of Computation” by Michael Sipser.
2. “Clean Code: A Handbook of Agile Software Craftsmanship” by Robert C. Martin.

Course Title: Introduction to Program Design

Course Description:

Introduction to Program Design provides foundational knowledge and practical skills for designing and implementing software programs. Students learn essential programming concepts, problem-solving techniques, and best practices. The course emphasizes clear program structure, readability, and modularity.

Learning Objectives:

By the end of this course, students will be able to:

1. **Understand Programming Fundamentals**:
    - Grasp basic programming concepts (variables, loops, conditionals).
    - Apply algorithmic thinking to solve problems.
2. **Design and Implement Programs**:
    - Develop program designs from specifications under various software paradigms/architectures.
    - Use analytical techniques to evaluate and compare different designs.
3. **Apply Data Structures**:
    - Adapt algorithms and data structures to new problems.
    - Understand the role of data structures in program design.

Topics Covered:

1. **Introduction to Programming**:
    - Overview of programming languages and their syntax.
    - Variables, data types, and expressions.
2. **Control Structures and Flow Control**:
    - Conditional statements (if, else).
    - Loops (while, for).
    - Error handling.
3. **Functions and Modular Design**:
    - Function definition and invocation.
    - Parameters and return values.
    - Scope and lifetime of variables.
4. **Data Structures**:
    - Arrays, lists, and dictionaries.
    - Strings and their manipulation.
5. **Algorithmic Thinking**:
    - Basic algorithmic techniques (brute force, divide and conquer).
    - Time and space complexity analysis.

Assessment:

- **Assignments**: Regular programming assignments to reinforce concepts.
- **Quizzes**: Assess understanding of topics covered.
- **Project**: Design and implement a small-scale program.
- **Participation**: Active engagement in class discussions and labs.

Recommended Reading:

1. “Introduction to the Theory of Computation” by Michael Sipser.
2. “Clean Code: A Handbook of Agile Software Craftsmanship” by Robert C. Martin.

Course Title: Introduction to Machine Organization and Assembly Language

Course Description:

Introduction to Machine Organization and Assembly Language provides foundational knowledge about computer systems’ organization and the usage of assembly language for optimization and control. Emphasis is placed on understanding low-level logic and problem-solving using assembly language. By the end of the course, students should be able to:

1. Identify major components of computer architecture and explain their purposes and interactions.
2. Simulate data representation, storage, and access in memory.
3. Understand the relationship between hardware architecture and instruction sets.
4. Explain the instruction execution cycle.
5. Compare high-level, assembly, and machine languages.

Learning Objectives:

Students will be able to:

1. Understand programming fundamentals and algorithmic thinking.
2. Write well-modularized assembly language programs, implementing decision structures and repetition.
3. Develop moderately complex assembly language subroutines and interface them with high-level languages.

Topics Covered:

1. **Data Representation**:
    - Binary numbers, hexadecimal integers.
    - Integer storage sizes, signed integers (sign-magnitude, 1’s complement, 2’s complement).
    - Floating-point representation.
2. **Intel 8088 (IAPX 88) Microprocessor**:
    - Architecture overview, registers (general purpose, pointer, segment).
    - Flag register and instruction groups (data movement, arithmetic, control).
3. **Implementing Assembly Code**:
    - Tools (debugger, linker, assembler).
    - Segmented memory model in IAPX 88.

- Address calculation, offset, segment physical address.

Assessment:

- **Assignments**: Regular programming assignments.
- **Quizzes**: Assess understanding of topics covered.
- **Project**: Design and implement assembly code.
- **Participation**: Active engagement in class discussions and labs.

Recommended Reading:

1. “Computer Organization & Architecture” by William Stallings.
2. “Using Assembly Language” by Allen L. Wyatt.

Course Title: Discrete Mathematics

Course Description:

Discrete Mathematics draws on areas of both computer science and mathematics. In this course, you’ll acquire skills in software engineering, combinatorial analysis, formal proof, and algorithmic analysis. The focus is on abstract problem-solving, logical reasoning, and practical applications.

Learning Objectives:

By the end of this course, students will be able to:

1. Understand fundamental mathematical concepts, including definitions, proofs, sets, and functions.
2. Apply discrete structures such as elementary number theory and graphs.
3. Explore counting techniques and discrete probability theory.

Topics Covered:

1. **Fundamental Concepts of Mathematics**:
   - Definitions and logical reasoning.
   - Sets and functions.
2. **Discrete Structures**:
   - Elementary number theory.
   - Graphs and their properties.
3. **Counting Techniques and Probability**:
   - Permutations and combinations.
   - Discrete probability models.

Assessment:

- **Assignments**: Regular problem-solving tasks.
- **Quizzes**: Assess understanding of concepts.
- **Projects**: Apply knowledge to practical problems.
- **Participation**: Engage in group discussions and projects.

Recommended Reading:

1. “Discrete Mathematics and Its Applications” by Kenneth H. Rosen.
2. “Concrete Mathematics: A Foundation for Computer Science” by Ronald L. Graham, Donald E. Knuth, and Oren Patashnik.

Course Title: Object-Oriented Design Concepts

Course Description:

Object-Oriented Design Concepts explores fundamental principles and practices related to object-oriented programming and design. Students delve into topics such as encapsulation, inheritance, polymorphism, and software design patterns. The course emphasizes practical application through case studies and hands-on coding in one or more object-oriented languages.

Learning Objectives:

By the end of this course, students will be able to:

1. **Knowledge and Understanding**:
    - Critically understand well-established object-oriented design patterns and software engineering practices.
    - Choose appropriate object-oriented design patterns for small to moderately sized programs.
2. **Subject Specific Skills**:
    - Evaluate the impact of program design decisions on technical, social, and management aspects of software.
    - Implement well-designed, well-structured, and well-documented code based on given software designs.

Topics Covered:

1. **Object-Oriented Concepts**:
    - Encapsulation, inheritance, and polymorphism.
    - Composition and object identity.
2. **Design Patterns**:
    - Commonly used design patterns (e.g., Singleton, Factory, Observer).
    - Examples of their practical application.

Assessment:

- **Assignments**: Regular problem-solving tasks.
- **Quizzes**: Assess understanding of concepts.
- **Project**: Design and implement object-oriented solutions.
- **Participation**: Engage in discussions and collaborative projects.

Recommended Reading:

1. “Design Patterns: Elements of Reusable Object-Oriented Software” by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides.

2. “Clean Code: A Handbook of Agile Software Craftsmanship” by Robert C. Martin.

Course Title: Accelerated Program Design

Course Description:

The **Accelerated Program Design** course focuses on essential principles and practices related to software development, design, and optimization. Students learn to create efficient and robust programs, emphasizing real-world applications and problem-solving. The course covers both high-level design concepts and low-level implementation details.

Learning Objectives:

By the end of this course, students will be able to:

1. **Software Development Fundamentals**:
    - Understand the software development life cycle.
    - Apply design principles to create maintainable and scalable code.
2. **Algorithmic Thinking and Problem-Solving**:
    - Analyze problems and design effective algorithms.
    - Implement solutions using appropriate programming languages.
3. **Optimization Techniques**:
    - Profile and optimize code for performance.
    - Explore trade-offs between time complexity and memory usage.

Topics Covered:

1. **Design Concepts**:
    - Object-oriented design (classes, inheritance, encapsulation).
    - Design patterns (e.g., Singleton, Factory, Observer).
2. **Programming Languages and Paradigms**:
    - Java, C++, Python, and scripting languages.
    - Functional vs. imperative programming.
3. **Software Engineering Practices**:
    - Version control (Git).
    - Documentation and code readability.

Assessment:

- **Assignments**: Practical coding tasks.
- **Projects**: Design and implement software solutions.
- **Quizzes**: Assess understanding of design principles.
- **Participation**: Engage in discussions and peer reviews.

Recommended Reading:

1. “Clean Code: A Handbook of Agile Software Craftsmanship” by Robert C. Martin.
2. “Design Patterns: Elements of Reusable Object-Oriented Software” by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides.

Course Title: Accelerated Object-Oriented Design Concepts

Course Description:

The **Accelerated Object-Oriented Design Concepts** course builds upon foundational knowledge in object-oriented programming. It assumes that students have significant prior experience in object-oriented programming. The course covers object-oriented design principles and data structures more deeply and at a faster pace than the standard introductory course (CS2102). Students will be expected to design, implement, test, debug, and critique programs both for correctness and adherence to good object-oriented design principles.

Learning Objectives:

By the end of this course, students will be able to:

1. **Solidify Object-Oriented Design Skills**:
    - Understand advanced design principles.
    - Apply design patterns effectively.
2. **Algorithmic Thinking and Implementation**:
    - Analyze problems and design efficient algorithms.
    - Implement solutions using object-oriented languages.

Topics Covered:

1. **Advanced Object-Oriented Concepts**:
    - Inheritance, encapsulation, and polymorphism.
    - Composition and object relationships.
2. **Design Patterns**:
    - Explore commonly used design patterns (e.g., Singleton, Factory, Observer).
    - Apply these patterns to real-world scenarios.

Assessment:

- **Assignments**: Practical coding tasks emphasizing design principles.
- **Projects**: Design and implement software solutions.
- **Quizzes**: Assess understanding of advanced design concepts.
- **Participation**: Engage in discussions and peer reviews.

Recommended Reading:

1. “Design Patterns: Elements of Reusable Object-Oriented Software” by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides.
2. “Clean Code: A Handbook of Agile Software Craftsmanship” by Robert C. Martin.

Course Title: Application Building with Object-Oriented Concepts

Course Description:

The **Application Building with Object-Oriented Concepts** course focuses on practical software development using object-oriented principles. Students learn how to design, build,

and optimize applications while applying key concepts such as encapsulation, inheritance, and polymorphism. The course emphasizes hands-on coding and real-world scenarios.

Learning Objectives:

By the end of this course, students will be able to:

1. **Solidify Object-Oriented Design Skills**:
    - Understand advanced design principles.
    - Apply design patterns effectively.
2. **Algorithmic Thinking and Implementation**:
    - Analyze problems and design efficient algorithms.
    - Implement solutions using object-oriented languages.

Topics Covered:

1. **Object-Oriented Concepts**:
    - Encapsulation, inheritance, and polymorphism.
    - Composition and object relationships.
2. **Software Development Practices**:
    - Effective program documentation.
    - Testing, readability, and modifiability.
3. **Application Building**:
    - Designing modular and maintainable code.
    - Implementing features using object-oriented techniques.

Assessment:

- **Assignments**: Practical coding tasks emphasizing design principles.
- **Projects**: Building complete applications.
- **Quizzes**: Assess understanding of advanced design concepts.
- **Participation**: Engage in discussions and peer reviews.

Recommended Reading:

1. “Design Patterns: Elements of Reusable Object-Oriented Software” by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides.
2. “Clean Code: A Handbook of Agile Software Craftsmanship” by Robert C. Martin.

Course Title: Algorithms and Data Structures

Course Description:

The **Algorithms and Data Structures** course introduces fundamental concepts related to efficient algorithms and essential data structures. Students learn how to analyze, design, and implement algorithms for solving computational problems. The course emphasizes both theoretical understanding and practical application.

Learning Objectives:

By the end of this course, students will be able to:

1. **Algorithmic Thinking**:
    - Understand algorithm design paradigms (e.g., divide and conquer, dynamic programming).
    - Analyze time and space complexity.
2. **Data Structures**:
    - Explore common data structures (e.g., arrays, linked lists, trees, graphs).
    - Implement and manipulate data structures efficiently.
3. **Problem-Solving Skills**:
    - Apply algorithms and data structures to real-world problems.
    - Optimize code for performance.

Topics Covered:

1. **Algorithm Design**:
    - Divide and conquer.
    - Greedy algorithms.
    - Dynamic programming.
2. **Data Structures**:
    - Arrays and linked lists.
    - Trees (binary trees, balanced trees).
    - Graphs (traversals, shortest paths).
3. **Sorting and Searching Algorithms**:
    - QuickSort, MergeSort.
    - Binary search.

Assessment:

- **Assignments**: Implement algorithms and data structures.
- **Exams**: Assess theoretical knowledge.
- **Projects**: Solve complex problems using algorithms.
- **Participation**: Engage in discussions and coding labs.

Recommended Reading:

1. **"Introduction to Algorithms"** by Thomas Cormen, Charles Leiserson, Ronald Rivest, and Clifford Stein.
2. "Algorithms" by S. Dasgupta, C.H. Papadimitriou, and U. V. Vazirani.

Course Title: System Programming Concepts

Course Description:

System Programming Concepts delves into the intricacies of low-level software development, focusing on system-level programming. Students learn how to interact with hardware, manage memory, and optimize code for performance. The course bridges the gap between high-level programming and the underlying computer architecture.

Learning Objectives:

By the end of this course, students will be able to:

1. **Understand System-Level Concepts**:
    - Explore operating system components (processes, threads, file systems).
    - Grasp memory management techniques.
2. **Low-Level Programming Skills**:
    - Write code that interacts with hardware (device drivers, interrupts).
    - Optimize performance using assembly language.

Topics Covered:

1. **Operating System Fundamentals**:
    - Process management (creation, scheduling, synchronization).
    - File I/O and system calls.
2. **Memory Management**:
    - Virtual memory, paging, and segmentation.
    - Memory allocation and deallocation.
3. **Low-Level Programming**:
    - Assembly language basics.
    - Interrupt handling and device drivers.

Assessment:

- **Assignments**: Implement system-level programs.
- **Projects**: Develop device drivers or system utilities.
- **Exams**: Assess theoretical knowledge.
- **Participation**: Engage in discussions and labs.

Recommended Reading:

1. “Operating System Concepts” by Abraham Silberschatz, Peter B. Galvin, and Greg Gagne.
2. “Professional Assembly Language” by Richard Blum.

Course Title: Computer Programming II

Course Description:

Computer Programming II builds upon the foundational programming concepts introduced in Computer Programming I. Students deepen their understanding of algorithm development, coding, debugging, and documentation. The course emphasizes structured programming, modularization, and problem-solving techniques.

Learning Objectives:

By the end of this course, students will be able to:

1. **Algorithm Development**:
    - Design efficient algorithms for solving computational problems.
    - Apply algorithmic thinking to real-world scenarios.
2. **Structured Programming**:
    - Understand control structures (loops, conditionals).
    - Modularize code using subroutines or subprograms.

3. **File Management**:
    - Read from and write to files.
    - Handle file I/O operations.

Topics Covered:

1. **Programming Steps and Concepts**:
    - Variables and constants.
    - Arithmetic operations.
    - Control structures (if, else, loops).
2. **Modularization and Subroutines**:
    - Creating reusable code blocks.
    - Function definition and invocation.
3. **File Handling**:
    - Reading and writing data to files.
    - Error handling for file operations.

Assessment:

- **Assignments**: Implement algorithms and modular programs.
- **Quizzes**: Assess understanding of programming concepts.
- **Projects**: Develop solutions to practical problems.
- **Participation**: Engage in coding labs and discussions.

Recommended Reading:

1. “Clean Code: A Handbook of Agile Software Craftsmanship” by Robert C. Martin.
2. “Introduction to the Theory of Computation” by Michael Sipser.

Course Title: Human-Computer Interaction

Course Description:

Human-Computer Interaction (HCI) focuses on designing interactive systems that effectively meet users’ needs. Students learn how to create user-friendly interfaces, conduct user research, and apply principles of user-centered design. The course bridges computer science, psychology, and design to ensure that digital products and services are usable, efficient, and enjoyable.

Learning Objectives:

By the end of this course, students will be able to:

1. **Understand HCI Concepts**:
    - Explore the history and importance of HCI.
    - Learn about user research and evaluation techniques.
2. **User-Centered Design**:
    - Apply principles of designing for users’ needs.
    - Create intuitive and efficient interfaces.

Topics Covered:

1. **Introduction to HCI**:
    - Historical context and evolution of HCI.
    - Importance of user-centered design.
2. **User Research Techniques**:
    - Surveys, interviews, and usability testing.
    - Gathering requirements from users.
3. **Design Principles and Prototyping**:
    - Interface design guidelines.
    - Rapid prototyping methods.

Assessment:

- **Assignments**: Design and evaluate interfaces.
- **Projects**: Apply HCI principles to real-world scenarios.
- **Exams**: Assess theoretical knowledge.
- **Participation**: Engage in discussions and labs.

Recommended Reading:

1. "The Design of Everyday Things" by Don Norman.
2. "Interaction Design: Beyond Human-Computer Interaction" by Jennifer Preece, Yvonne Rogers, and Helen Sharp.

Course Title: Social Implications for Information Processing

Course Description:

The **Social Implications for Information Processing** course explores the ethical, legal, and societal aspects of computing. Students learn to analyze the impact of technology on individuals, communities, and society at large. The course emphasizes critical thinking, responsible decision-making, and awareness of the broader implications of computer science.

Learning Objectives:

By the end of this course, students will be able to:

1. **Ethical Awareness**:
    - Identify ethical issues related to computer science in real-world scenarios.
    - Evaluate actions based on computer science professional ethics.
2. **Legal Considerations**:
    - Recognize situations with legal implications (e.g., intellectual property, privacy).
    - Understand relevant legal principles.
3. **Social and Cultural Context**:
    - Explore the social impact of technology.
    - Discuss economic, diversity, and human rights issues.

Topics Covered:

1. **Ethics and Professional Responsibility**:
    - History of computing ethics.
    - Intellectual property and privacy.

2. **Legal Aspects**:
    - Legal principles related to computer science.
    - Liability and safety-critical systems.
3. **Social Implications**:
    - Economic impact of technology.
    - Diversity, inclusion, and bias.

Assessment:

- **Weekly Homework**: Research-based assignments.
- **Oral Presentation**: Deliver a professional-quality talk on a relevant topic.
- **Research Paper**: Analyze an issue related to social, legal, or ethical implications.
- **Class Discussions**: Engage in debates based on legal or ethical principles.

Recommended Reading:

1. Michael J. Quinn, “Ethics for the Information Age” (4th edition, Addison Wesley, 2011).
2. Sara Baase, “A Gift of Fire: Social, Legal, and Ethical Issues for Computing and the Internet” (3rd edition, Pearson/Prentice Hall, 2007)

Course Title: Foundations of Computer Science

Course Description:

The **Foundations of Computer Science** course provides a solid introduction to fundamental concepts in computer science. It covers essential topics that serve as building blocks for more advanced studies. Students gain a strong theoretical foundation and practical problem-solving skills.

Learning Objectives:

By the end of this course, students will be able to:

1. **Understand Core Concepts**:
    - Grasp fundamental principles such as sets, functions, and relations.
    - Explore graph theory, trees, and mathematical induction.
2. **Problem-Solving Skills**:
    - Apply logical reasoning to solve computational problems.
    - Analyze and evaluate solutions.

Topics Covered:

1. **Sets and Relations**:
    - Set theory, set operations, and cardinality.
    - Relations, equivalence relations, and partial orders.
2. **Graphs and Trees**:
    - Graph theory (vertices, edges, paths, cycles).
    - Tree structures (binary trees, spanning trees).
3. **Mathematical Induction**:
    - Proof by induction.
    - Recursive algorithms.

Assessment:

- **Assignments**: Problem-solving exercises.
- **Quizzes**: Assess understanding of theoretical concepts.
- **Projects**: Apply concepts to practical scenarios.
- **Participation**: Engage in class discussions and labs.

Recommended Reading:

1. "Discrete Mathematics and Its Applications" by Kenneth H. Rosen.
2. "Introduction to the Theory of Computation" by Michael Sipser.

Course Title: Database Systems I

Course Description:

Database Systems I introduce fundamental concepts related to database management systems (DBMSs). Students learn about organizing and retrieving large amounts of structured data efficiently. The course covers the Relational Model, SQL (Structured Query Language), and how to design and query databases. Additionally, it explores accessing databases from procedural programming languages like Java or C++.

Learning Objectives:

By the end of this course, students will be able to:

1. **Design Efficient Databases**:
    - Create appropriate and efficient database designs for various applications.
    - Understand schema definition and logical database design.
2. **SQL Implementation**:
    - Use SQL to implement database designs.
    - Translate informal queries into formal notations.
3. **Normalization and Integrity Constraints**:
    - Apply normalization theory to resolve database anomalies.
    - Identify and express relevant integrity constraints.

Topics Covered:

1. **Overview of Databases and DBMSs**:
    - Importance of databases.
    - Relational Model and Relational DBMSs.
2. **SQL Declarative Programming Language**:
    - Writing SQL commands.
    - Creating and indexing tables.
    - Formatting query results into reports.
3. **Theoretical Underpinnings**:
    - Normalization theory.
    - Relational Algebra and Relational Calculus.
4. **Security and Object-Relation Mapping**:
    - Identifying security threats.
    - Control measures for database security.

Assessment:

- **Assignments**: Problem-solving exercises related to database design and SQL.
- **Projects**: Implementing and querying databases.
- **Exams**: Assess theoretical knowledge.
- **Participation**: Engage in class discussions and labs.

Recommended Reading:

1. “Database System Concepts” by Abraham Silberschatz, Henry F. Korth, and S. Sudarshan.
2. “SQL Performance Explained” by Markus Winand.

Course Title: Computer Networks

Course Description:

The **Computer Networks** course focuses on the infrastructure that powers the movement of data across the Internet. Students gain an in-depth understanding of computer network design, implementation, and optimization. The course covers cutting-edge technologies and prepares students for careers in networking and related fields.

Learning Objectives:

By the end of this course, students will be able to:

1. **Network Design and Implementation**:
    - Understand principles of network architecture.
    - Design and configure network infrastructures.
2. **Network Optimization and Performance**:
    - Analyze and evaluate network performance.
    - Implement strategies for efficient data transmission.

Topics Covered:

1. **Introduction to Networking**:
    - Overview of network types (LAN, WAN, MAN).
    - Network protocols (TCP/IP, OSI model).
2. **Network Components and Devices**:
    - Routers, switches, hubs, and gateways.
    - Network topologies (star, bus, ring).
3. **Network Security and Management**:
    - Security protocols (firewalls, VPNs).
    - Network monitoring and troubleshooting.

Assessment:

- **Assignments**: Network design and configuration tasks.
- **Projects**: Implementing and optimizing network solutions.
- **Exams**: Assess theoretical knowledge.
- **Participation**: Engage in discussions and labs.

Recommended Reading:

1. “Computer Networking: A Top-Down Approach” by James F. Kurose and Keith W. Ross.
2. “TCP/IP Illustrated, Volume 1: The Protocols” by W. Richard Stevens.

Course Title: Software Engineering

Course Description:

The **Software Engineering** course focuses on principles and practices related to designing, developing, and maintaining high-quality software systems. Students learn about software development methodologies, project management, and best practices. The course emphasizes both theoretical understanding and practical application.

Learning Objectives:

By the end of this course, students will be able to:

1. **Software Development Lifecycle**:
    - Understand the entire software development process, including requirements analysis, design, implementation, testing, and maintenance.
    - Apply software engineering principles to real-world projects.
2. **Software Design and Architecture**:
    - Learn about software design patterns, modularization, and architectural styles.
    - Create well-structured and maintainable software designs.
3. **Quality Assurance and Testing**:
    - Implement effective testing strategies (unit testing, integration testing, etc.).
    - Ensure software quality through rigorous testing and debugging.

Topics Covered:

1. **Software Development Methodologies**:
    - Waterfall, Agile, Scrum, and other development models.
    - Iterative and incremental development.
2. **Requirements Engineering**:
    - Gathering and analyzing user requirements.
    - Creating clear and concise software specifications.
3. **Software Design Patterns**:
    - Singleton, Factory, Observer, and other common patterns.
    - Applying design patterns to solve recurring problems.
4. **Software Testing and Quality Assurance**:
    - Test planning, test cases, and test automation.
    - Code reviews and quality metrics.

Assessment:

- **Assignments**: Design and implement software projects.
- **Projects**: Apply software engineering principles to real-world scenarios.
- **Exams**: Assess theoretical knowledge.
- **Participation**: Engage in discussions and labs.

Recommended Reading:

1. “Clean Code: A Handbook of Agile Software Craftsmanship” by Robert C. Martin.
2. “Design Patterns: Elements of Reusable Object-Oriented Software” by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides.

Course Title: Theory of Automata

Course Description:

The **Theory of Automata** course explores fundamental concepts related to formal languages, automata, and computational models. Students learn about the theoretical foundations of computation, including finite automata, context-free grammars, Turing machines, and decidability. The course emphasizes both theoretical understanding and practical applications.

Learning Objectives:

By the end of this course, students will be able to:

1. **Formal Language Theory**:
    - Understand the hierarchy of formal languages (regular, context-free, context-sensitive, recursively enumerable).
    - Analyze language-defining mechanisms such as finite automata and context-free grammars.
2. **Automata and Computation**:
    - Study deterministic and nondeterministic automata.
    - Explore the equivalence of different computational models (e.g., pushdown automata and context-free grammars).
3. **Decidability and Complexity**:
    - Learn about undecidable problems and the limits of computation.
    - Understand complexity classes (e.g., P, NP, NP-complete).

Topics Covered:

1. **Finite Automata and Regular Languages**:
    - Deterministic and nondeterministic finite automata.
    - Regular expressions and regular languages.
2. **Context-Free Grammars and Pushdown Automata**:
    - Context-free languages and context-free grammars.
    - Parsing techniques and derivation trees.
3. **Turing Machines and Computability**:
    - Turing machines and their variants.
    - Decidability and undecidability results.
4. **Complexity Theory**:
    - Time and space complexity classes.
    - NP-completeness and intractable problems.

Assessment:

- **Assignments**: Problem-solving exercises related to formal languages and automata.
- **Projects**: Implementing automata or solving theoretical problems.

- **Exams**: Assess theoretical knowledge.
- **Participation**: Engage in discussions and labs.

Recommended Reading:

1. “Introduction to the Theory of Computation” by Michael Sipser.
2. “Automata Theory, Languages, and Computation” by Hopcroft, Motwani, and Ullman.

Course Title: Numerical Methods for Linear and Non-Linear Systems

Course Description:

The **Numerical Methods for Linear and Non-Linear Systems** course focuses on algorithms and techniques for solving mathematical problems encountered in scientific and engineering applications. Students learn how to approximate solutions to linear and non-linear equations, optimize functions, and analyze numerical errors. The course emphasizes both theoretical understanding and practical implementation.

Learning Objectives:

By the end of this course, students will be able to:

1. **Numerical Algorithms**:
    - Understand fundamental numerical methods for solving linear and non-linear systems.
    - Apply these methods to real-world problems.
2. **Error Analysis**:
    - Analyze and quantify numerical errors.
    - Implement strategies to minimize error propagation.

Topics Covered:

1. **Linear Systems**:
    - Gaussian elimination and LU decomposition.
    - Iterative methods (Jacobi, Gauss-Seidel).
2. **Non-Linear Systems**:
    - Newton-Raphson method.
    - Bisection and secant methods.
3. **Optimization Techniques**:
    - Gradient-based optimization.
    - Constrained optimization.

Assessment:

- **Assignments**: Implement numerical algorithms.
- **Projects**: Solve practical problems using numerical methods.
- **Exams**: Assess theoretical knowledge.
- **Participation**: Engage in discussions and labs.

Recommended Reading:

1. Uri Ascher & Chen Greif: “A First Course in Numerical Methods.” SIAM, 2011
2. W. Dahmen & A. Reusken: “Numerik für Ingenieure und Naturwissenschaftler.” Springer, 2006

Course Title: Numerical Methods for Calculus and Differential Equations

Course Description:

The **Numerical Methods for Calculus and Differential Equations** course focuses on algorithms and techniques for approximating solutions to mathematical problems encountered in scientific and engineering applications. Students learn how to apply numerical methods to solve differential equations, optimize functions, and analyze numerical errors. The course emphasizes both theoretical understanding and practical implementation.

Learning Objectives:

By the end of this course, students will be able to:

1. **Numerical Algorithms**:
    - Understand fundamental numerical methods for solving differential equations.
    - Apply these methods to real-world problems.
2. **Error Analysis**:
    - Analyze and quantify numerical errors.
    - Implement strategies to minimize error propagation.

Topics Covered:

1. **Approximation Techniques**:
    - Euler’s method for solving ordinary differential equations.
    - Higher-order methods (e.g., Runge-Kutta methods).
2. **Numerical Optimization**:
    - Gradient-based optimization algorithms.
    - Constrained optimization techniques.
3. **Partial Differential Equations**:
    - Finite difference methods for solving partial differential equations.
    - Boundary value problems and initial value problems.

Assessment:

- **Assignments**: Implement numerical algorithms for differential equations.
- **Projects**: Solve practical problems using numerical methods.
- **Exams**: Assess theoretical knowledge.
- **Participation**: Engage in discussions and labs.

Recommended Reading:

1. Uri Ascher & Chen Greif: “A First Course in Numerical Methods.” SIAM, 2011
2. W. Dahmen & A. Reusken: “Numerik für Ingenieure und Naturwissenschaftler.” Springer, 2006

Course Title: Artificial Intelligence for Interactive Media and Games

Course Description:

The **Artificial Intelligence for Interactive Media and Games** course focuses on the application of AI techniques to create intelligent and interactive systems for media, entertainment, and gaming. Students learn how to design, implement, and optimize AI algorithms for game development, virtual reality, and interactive storytelling. The course emphasizes both theoretical understanding and practical implementation.

Learning Objectives:

By the end of this course, students will be able to:

1. **AI Fundamentals**:
    - Understand the core concepts of artificial intelligence (machine learning, natural language processing, computer vision).
    - Apply AI techniques to interactive media and game development.
2. **Game AI Design and Implementation**:
    - Create intelligent agents for non-player characters (NPCs).
    - Implement pathfinding, decision-making, and behavior trees.
3. **Interactive Storytelling and Virtual Reality**:
    - Explore AI-driven narrative generation.
    - Apply AI techniques to enhance user experiences in virtual environments.

Topics Covered:

1. **Introduction to Game AI**:
    - Overview of game AI techniques.
    - Role of AI in interactive media and games.
2. **Pathfinding and Navigation**:
    - A* algorithm, Dijkstra's algorithm.
    - Navigation meshes and waypoints.
3. **Decision-Making and Behavior Trees**:
    - Finite state machines (FSMs).
    - Hierarchical task networks (HTNs).
4. **AI in Virtual Reality and Interactive Storytelling**:
    - Procedural content generation.
    - AI-driven narrative design.

Assessment:

- **Assignments**: Implement game AI algorithms.
- **Projects**: Develop intelligent NPCs or interactive experiences.
- **Exams**: Assess theoretical knowledge.
- **Participation**: Engage in discussions and labs.

Recommended Reading:

1. Ian Millington: "Artificial Intelligence for Games" (3rd edition, CRC Press, 2019)
2. Steve Rabin (Editor): "AI Game Programming Wisdom" (Charles River Media, 2002)

Course Title: Biovisualization

Course Description:

The **Biovisualization** course focuses on the application of visualization techniques to biological data and scientific communication. Students learn how to create effective visual representations of complex biological information, such as molecular structures, genetic sequences, and biological pathways. The course emphasizes both theoretical understanding and practical implementation.

Learning Objectives:

By the end of this course, students will be able to:

1. **Understand Biological Data**:
    - Grasp fundamental concepts in biology, genetics, and bioinformatics.
    - Analyze biological datasets and identify relevant features.
2. **Visualization Techniques**:
    - Learn about data visualization principles and tools.
    - Apply visualization techniques to represent biological data.
3. **Effective Communication**:
    - Create compelling visualizations for scientific communication.
    - Understand the ethical considerations in biovisualization.

Topics Covered:

1. **Introduction to Biovisualization**:
    - Overview of biological data types (DNA, proteins, pathways).
    - Challenges in visualizing complex biological structures.
2. **Visualization Tools and Libraries**:
    - Using Python libraries (Matplotlib, Seaborn, Plotly) for biovisualization.
    - Interactive visualizations and 3D modeling.
3. **Case Studies and Projects**:
    - Visualizing protein structures.
    - Genome mapping and phylogenetic trees.

Assessment:

- **Assignments**: Create biovisualizations based on real-world datasets.
- **Projects**: Develop interactive visualizations for biological research.
- **Exams**: Assess theoretical knowledge.
- **Participation**: Engage in discussions and labs.

Recommended Reading:

1. Tamara Munzner: “Visualization Analysis and Design” (CRC Press, 2014)
2. Bang Wong: “Points of View: Colorful Data Visualization” (Nature Methods, 2010)

Course Title: Application Development and Emerging Technologies I

Course Description:

The **Application Development and Emerging Technologies I** course focuses on the development of software applications using web and emerging technologies. It explores potential benefits from these technologies and assesses their feasibility for implementation in various systems. Emphasis is placed on requirements management, interface design, usability, and ethical and legal considerations.

Learning Objectives:

By the end of this course, students will be able to:

1. **Understand Web and Emerging Technologies**:
    - Grasp fundamental concepts related to web development and emerging tech.
    - Analyze the impact of these technologies on software systems.
2. **Software Design and Implementation**:
    - Apply principles of interface design and usability.
    - Develop software applications using web technologies.
3. **Ethical and Legal Considerations**:
    - Understand the ethical implications of technology adoption.
    - Comply with legal requirements in software development.

Topics Covered:

1. **Introduction to Web Technologies**:
    - Overview of web development tools, languages, and frameworks.
    - Client-server architecture and RESTful APIs.
2. **User Interface Design and Usability**:
    - Principles of effective interface design.
    - User experience (UX) considerations.
3. **Emerging Technologies**:
    - Exploring trends such as blockchain, IoT, and AI.
    - Feasibility assessment for system integration.

Assessment:

- **Assignments**: Implement web-based applications.
- **Projects**: Develop software prototypes using emerging technologies.
- **Exams**: Assess theoretical knowledge.
- **Participation**: Engage in discussions and labs.

Recommended Reading:

1. Uri Ascher & Chen Greif: “A First Course in Numerical Methods” (SIAM, 2011)
2. W. Dahmen & A. Reusken: “Numerik für Ingenieure und Naturwissenschaftler” (Springer, 2006)

Course Title: Industrial Attachment

Course Description:

The **Industrial Attachment** provides students with practical experience in a professional work environment related to computer science. During this period, students apply their theoretical knowledge and skills gained throughout their studies to real-world projects. The attachment aims to bridge the gap between academic learning and industry practices.

Learning Objectives:

By the end of the Industrial Attachment, students will be able to:

1. **Apply Theoretical Knowledge**:
    - Transfer concepts learned in the classroom to practical situations.
    - Gain hands-on experience in a professional setting.
2. **Professional Development**:
    - Develop workplace skills such as communication, teamwork, and time management.
    - Understand industry practices, standards, and ethics.

Topics Covered:

1. **Workplace Experience**:
    - Exposure to daily tasks and responsibilities in a professional environment.
    - Interaction with colleagues, supervisors, and clients.
2. **Project Work**:
    - Participation in ongoing projects or specific assignments.
    - Application of computer science skills to solve real-world problems.

Assessment:

- **Supervisor Evaluation**: Assessment by the industrial supervisor based on the student's performance during the attachment.
- **Student Report**: Submission of a report summarizing the experience, challenges, and lessons learned during the attachment.

Recommended Reading:

- No specific reading materials are required for the Industrial Attachment. However, students are encouraged to keep a journal of their experiences and reflections during this period.

Course Title: Webware: Computational Technology for Network Information Systems

Course Description:

The **Webware: Computational Technology for Network Information Systems** course explores the computational aspects of network information systems as embodied by the World Wide Web (WWW). Students will delve into various topics related to web development, programming languages, and network architecture. The course emphasizes practical skills in

designing web-based human-computer interfaces, understanding high-level network protocols, and addressing network security issues. Students will also complete a substantial software project related to web technologies.

Learning Objectives:

By the end of this course, students will be able to:

1. **Understand Web Technologies**:
    - Grasp fundamental concepts related to web development, languages for document design, and programming languages for executable content.
    - Explore scripting languages and their role in web-based systems.
2. **Design and Implement Web-Based Interfaces**:
    - Apply principles of interface design to create user-friendly web applications.
    - Understand client/server network architecture models.
3. **Network Security and Resource Discovery**:
    - Analyze network security issues specific to web applications.
    - Explore high-level network protocols such as HTTP.
    - Learn about resource discovery mechanisms on the WWW.

Topics Covered:

1. **Languages for Document Design**:
    - HTML, CSS, and JavaScript.
    - Creating web pages and user interfaces.
2. **Programming Languages for Executable Content**:
    - Server-side scripting languages (e.g., PHP, Python, Ruby).
    - Dynamic content generation.
3. **Network Resource Discovery**:
    - Understanding URLs and URI schemes.
    - Search engines and web crawlers.

Assessment:

- **Substantial Software Project**: Students will complete a significant software project related to web technologies (e.g., Java-based user interface, HTML/CGI-based information system, or WWW search mechanisms).
- **Exams**: Assess theoretical knowledge.
- **Participation**: Engage in discussions and labs.

Recommended Reading:

1. Uri Ascher & Chen Greif: “A First Course in Numerical Methods” (SIAM, 2011)
2. W. Dahmen & A. Reusken: “Numerik für Ingenieure und Naturwissenschaftler” (Springer, 2006)

C**ourse Title: Tools and Techniques in Computer Science**

Course Description: The **Tools and Techniques in Computer Science** course provides students with practical skills and knowledge related to essential tools, methodologies, and techniques used in computer science. Students will learn how to effectively use software

development tools, analyze algorithms, and apply problem-solving techniques. The course emphasizes hands-on experience and critical thinking.

Learning Objectives:

By the end of this course, students will be able to:

1. **Master Software Development Tools**:
    - Understand and utilize integrated development environments (IDEs), version control systems (e.g., Git), and debugging tools.
    - Apply best practices for code organization, documentation, and collaboration.
2. **Analyze Algorithms and Data Structures**:
    - Evaluate time and space complexity of algorithms.
    - Implement and optimize common data structures (e.g., arrays, linked lists, trees).
3. **Problem-Solving Techniques**:
    - Learn systematic approaches to problem-solving (e.g., divide and conquer, dynamic programming).
    - Apply these techniques to real-world programming challenges.

Topics Covered:

1. **Software Development Tools**:
    - IDEs (e.g., Visual Studio Code, IntelliJ IDEA).
    - Version control (Git, GitHub).
    - Debugging and profiling tools.
2. **Algorithm Analysis and Data Structures**:
    - Big O notation and algorithm complexity.
    - Arrays, stacks, queues, and hash tables.
3. **Problem-Solving Strategies**:
    - Recursion, dynamic programming, and greedy algorithms.
    - Graph algorithms (e.g., breadth-first search, depth-first search).

Assessment:

- **Assignments**: Implement algorithms, analyze code, and use development tools.
- **Projects**: Apply problem-solving techniques to real-world scenarios.
- **Exams**: Assess theoretical knowledge.
- **Participation**: Engage in discussions and labs.

Recommended Reading:

1. Jon Bentley: “Programming Pearls” (Addison-Wesley Professional, 1999)
2. Thomas H. Cormen et al.: “Introduction to Algorithms” (MIT Press, 2009)

Course Title: Database Systems II

Course Description:

The **Database Systems II** course builds upon the foundational concepts covered in Database Systems I. It delves deeper into advanced topics related to database management systems

(DBMSs) and explores more complex scenarios. Students will learn about query optimization, transaction management, distributed databases, and emerging trends in database technology.

Learning Objectives:

By the end of this course, students will be able to:

1. **Advanced Query Optimization**:
    - Understand query execution plans and cost-based optimization.
    - Optimize complex SQL queries.
2. **Transaction Management**:
    - Learn about ACID properties (Atomicity, Consistency, Isolation, Durability).
    - Implement multi-step transactions and handle concurrency control.
3. **Distributed Databases**:
    - Explore distributed database architectures.
    - Understand data fragmentation, replication, and consistency models.

Topics Covered:

1. **Query Optimization**:
    - Query execution plans.
    - Cost-based optimization.
2. **Transaction Management**:
    - ACID properties.
    - Concurrency control mechanisms (locking, timestamps).
3. **Distributed Databases**:
    - Distributed database design.
    - Replication and consistency models.

Assessment:

- **Assignments**: Optimize queries, design distributed database schemas.
- **Projects**: Implement transaction management and distributed database solutions.
- **Exams**: Assess theoretical knowledge.
- **Participation**: Engage in discussions and labs.

Recommended Reading:

1. Raghu Ramakrishnan and Johannes Gehrke: “Database Management Systems” (3rd edition, McGraw-Hill, 2002)
2. Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom: “Database Systems: The Complete Book” (2nd edition, Pearson, 2008)

Course Title: Data Mining and Knowledge Discovery in Databases

Course Description:

The **Data Mining and Knowledge Discovery in Databases** course explores techniques and methodologies for extracting valuable patterns, knowledge, and insights from large datasets. Students will learn how to apply data mining algorithms, evaluate their performance, and

interpret the results. The course emphasizes both theoretical understanding and practical application.

Learning Objectives:

By the end of this course, students will be able to:

1. **Understand Data Mining Concepts**:
   - Grasp fundamental concepts related to data mining, machine learning, and pattern recognition.
   - Explore the process of knowledge discovery from databases.
2. **Apply Data Mining Techniques**:
   - Learn about various data mining algorithms (classification, clustering, association rules, etc.).
   - Apply these techniques to real-world datasets.
3. **Evaluate and Interpret Results**:
   - Assess the quality of data mining models.
   - Interpret patterns and trends discovered in the data.

Topics Covered:

1. **Introduction to Data Mining**:
   - Overview of data mining process and tasks.
   - Data preprocessing and feature selection.
2. **Supervised Learning Algorithms**:
   - Decision trees, neural networks, and support vector machines.
   - Model evaluation and validation.
3. **Unsupervised Learning Techniques**:
   - Clustering algorithms (k-means, hierarchical clustering).
   - Association rule mining.

Assessment:

- **Assignments**: Implement data mining algorithms, analyze results.
- **Projects**: Apply data mining techniques to solve real-world problems.
- **Exams**: Assess theoretical knowledge.
- **Participation**: Engage in discussions and labs.

Recommended Reading:

1. Jiawei Han, Micheline Kamber, and Jian Pei: “Data Mining: Concepts and Techniques” (3rd edition, Morgan Kaufmann, 2011)
2. Charu Aggarwal: “Data Mining: The Textbook” (Online version)

Course Title: Emerging Technology

Course Description:

The **Emerging Technology** course explores cutting-edge trends, innovations, and technologies in the field of computer science. Students will learn about the latest advancements, their impact on various industries, and how to adapt to rapidly changing

technological landscapes. The course emphasizes both theoretical understanding and practical application.

Learning Objectives:

By the end of this course, students will be able to:

1. **Stay Current with Technological Trends**:
    - Grasp fundamental concepts related to emerging technologies.
    - Understand the implications of these technologies on society, business, and daily life.
2. **Evaluate and Apply New Technologies**:
    - Learn how to assess the feasibility and potential of emerging tools and frameworks.
    - Explore practical use cases and implementation strategies.

Topics Covered:

1. **Blockchain and Cryptocurrencies**:
    - Understanding decentralized ledgers and smart contracts.
    - Exploring applications beyond cryptocurrencies.
2. **Artificial Intelligence and Machine Learning**:
    - Deep learning, neural networks, and natural language processing.
    - Practical applications in data analysis and automation.
3. **Internet of Things (IoT)**:
    - Connecting devices, sensors, and actuators.
    - Security and privacy considerations.

Assessment:

- **Assignments**: Research and analyze emerging technologies.
- **Projects**: Develop prototypes or proof-of-concept applications.
- **Exams**: Assess theoretical knowledge.
- **Participation**: Engage in discussions and labs.

Recommended Reading:

1. **"The Fourth Industrial Revolution"** by Klaus Schwab
2. **"Machine Learning: A Probabilistic Perspective"** by Kevin P. Murphy

Course Title: Software Security Engineering

Course Description:

Software Security Engineering focuses on identifying, preventing, and mitigating security vulnerabilities in software systems. Students learn about secure coding practices, threat modeling, and risk assessment. The course emphasizes practical skills to create robust and secure software.

Learning Objectives:

1. Understand common security threats and vulnerabilities.
2. Apply secure coding principles to prevent common security flaws.
3. Analyze and assess software security risks.
4. Design and implement secure software architectures.
5. Evaluate security tools and techniques.

Topics Covered:

1. **Introduction to Software Security:**
    - Overview of security concepts, confidentiality, integrity, and availability.
    - Threat modeling and risk assessment.
2. **Secure Coding Practices:**
    - Input validation and output encoding.
    - Authentication and authorization mechanisms.
    - Secure session management.
3. **Common Vulnerabilities and Mitigations:**
    - Cross-Site Scripting (XSS), SQL injection, and buffer overflows.
    - Security headers and Content Security Policy (CSP).
4. **Secure Software Development Lifecycle (SDLC):**
    - Integrating security into the SDLC.
    - Code reviews and static analysis tools.
5. **Security Testing and Assessment:**
    - Dynamic application security testing (DAST) and penetration testing.
    - Secure code review techniques.

Assessment:

- Assignments: Practical exercises on secure coding practices.
- Midterm Exam: Assessing theoretical knowledge.
- Final Project: Design and implement a secure software application.
- Participation: Active engagement in class discussions and labs.

Recommended Reading:

1. “The Web Application Hacker’s Handbook” by Dafydd Stuttard and Marcus Pinto.
2. “Secure Coding in C and C++” by Robert C. Seacord.
3. OWASP (Open Web Application Security Project) resources.

Course Title: Distributed Computer Systems

Course Description:

This course explores the design and implementation of distributed systems. Students learn about abstractions, networking, fault tolerance, and security in the context of distributed computing. The course emphasizes practical skills to create scalable and reliable systems.

Learning Objectives:

1. Understand the principles of distributed computing.
2. Design and implement networked applications.
3. Explore fault tolerance mechanisms.

4. Analyze trade-offs between consistency and availability.

Topics Covered:

1. **Distributed System Models:**
    - Client-server architecture, peer-to-peer networks.
    - Middleware and communication protocols.
2. **Network Programming:**
    - Socket programming, remote procedure calls (RPC).
    - Message-passing interfaces (MPI).
3. **Distributed File Systems:**
    - NFS (Network File System), HDFS (Hadoop Distributed File System).
    - Consistency models and replication.
4. **Security in Distributed Systems:**
    - Authentication, encryption, and access control.
    - Threats and countermeasures.

Assessment:

- Assignments: Implementing distributed algorithms and protocols.
- Midterm Exam: Assessing theoretical knowledge.
- Group Project: Design and deploy a distributed application.
- Participation: Active engagement in discussions and labs.

Recommended Reading:

1. "Distributed Systems: Principles and Paradigms" by Andrew S. Tanenbaum and Maarten van Steen.
2. "Designing Data-Intensive Applications" by Martin Kleppmann.
3. Research papers on distributed systems from conferences like USENIX, ACM, and IEEE.

Course Title: Mobile and Ubiquitous Computing

Course Description:

Mobile and Ubiquitous Computing explores the design, development, and deployment of software systems that operate across diverse devices, platforms, and environments. Students learn to create applications for mobile devices, wearables, and IoT (Internet of Things) devices. The course emphasizes practical skills and understanding the unique challenges of mobile and ubiquitous computing.

Learning Objectives:

1. **Mobile Application Development:**
    - Understand mobile platforms (iOS, Android, etc.).
    - Develop responsive and user-friendly mobile apps.
2. **Ubiquitous Computing Concepts:**
    - Study context-aware computing.
    - Explore location-based services.
3. **User Experience (UX) Design for Mobile:**

- Design intuitive interfaces.
- Optimize user interactions on small screens.

Topics Covered:

1. **Mobile Development Fundamentals:**
   - Mobile app lifecycle, UI components, and event handling.
   - Mobile databases and data synchronization.
2. **Cross-Platform Development:**
   - React Native, Flutter, or Xamarin.
   - Code sharing and platform-specific features.
3. **Wearable Technologies:**
   - Smartwatches, fitness trackers, and augmented reality glasses.
   - Developing apps for wearables.
4. **Security and Privacy in Mobile Apps:**
   - Authentication, encryption, and secure data storage.
   - Privacy implications of location services.

Assessment:

- **Mobile App Projects:** Develop mobile apps individually or in teams.
- **Quizzes and Exams:** Assess theoretical knowledge.
- **User Experience (UX) Evaluation:** Evaluate app usability.
- **Class Participation:** Engage in discussions and labs.

Recommended Reading:

1. "Mobile Design and Development" by Brian Fling.
2. "Programming Mobile Applications for Android" by Joseph Annuzzi Jr., Lauren Darcey, and Shane Conder.
3. Research papers on mobile computing from conferences like ACM MobiSys and IEEE PerCom.

Course Title: Graphics Computing and Multimedia Technology

Course Description:

Graphics Computing and Multimedia Technology explores the creation, manipulation, and presentation of visual content. Students learn about graphics programming, animation, 3D modeling, and multimedia design. The course emphasizes practical skills in creating compelling visual experiences.

Learning Objectives:

1. **Graphics Programming Fundamentals:**
   - Understand graphics APIs (OpenGL, DirectX).
   - Implement 2D and 3D graphics algorithms.
2. **Multimedia Design Principles:**
   - Study color theory, typography, and layout.
   - Explore multimedia storytelling.
3. **3D Modeling and Animation:**

- Create 3D models using software like Blender or Maya.
- Animate objects and characters.

Topics Covered:

1. **Computer Graphics Basics:**
   - Raster vs. vector graphics.
   - Transformations, shading, and rendering.
2. **Multimedia Authoring Tools:**
   - Adobe Creative Suite (Photoshop, Illustrator, Premiere).
   - Interactive multimedia development.
3. **Virtual Reality (VR) and Augmented Reality (AR):**
   - VR environments, AR applications.
   - User interaction in immersive experiences.
4. **Digital Audio and Video:**
   - Compression techniques.
   - Editing and post-production.

Assessment:

- **Graphics Projects:** Create visual assets (2D/3D graphics, animations).
- **Design Portfolio:** Showcase multimedia projects.
- **Exams and Quizzes:** Assess theoretical knowledge.
- **Class Participation:** Engage in discussions and labs.

Recommended Reading:

1. "Computer Graphics: Principles and Practice" by John F. Hughes, Andries van Dam, Morgan McGuire, and David F. Sklar.
2. "The Animator's Survival Kit" by Richard Williams.
3. Research papers on graphics and multimedia from conferences like SIGGRAPH and ACM Multimedia.

Course Title: Technopreneurship

Course Description:

Technopreneurship explores the intersection of technology and entrepreneurship. Students learn how to identify business opportunities, create innovative solutions, and launch tech-based ventures. The course emphasizes practical skills in product development, market analysis, and startup management.

Learning Objectives:

1. **Entrepreneurial Mindset:**
   - Understand the entrepreneurial ecosystem.
   - Cultivate creativity and risk-taking.
2. **Idea Generation and Validation:**
   - Brainstorming and ideation.
   - Market research and feasibility analysis.
3. **Product Development and Prototyping:**

- Design thinking and rapid prototyping.
- Iterative development cycles.

Topics Covered:

1. **Technology Trends and Disruption:**
   - Identifying emerging technologies.
   - Impact of AI, blockchain, and IoT on entrepreneurship.
2. **Business Models and Revenue Streams:**
   - Subscription models, freemium, and licensing.
   - Monetizing digital products.
3. **Startup Fundamentals:**
   - Legal considerations (IP, contracts).
   - Pitching to investors and securing funding.
4. **Scaling and Growth Strategies:**
   - Scaling tech startups.
   - Marketing and customer acquisition.

Assessment:

- **Business Plan:** Develop a tech-focused business idea.
- **Pitch Presentation:** Present your startup concept.
- **Case Studies:** Analyze successful tech ventures.
- **Class Participation:** Engage in discussions and workshops.

Recommended Reading:

1. "The Lean Startup" by Eric Ries.
2. "Zero to One" by Peter Thiel.
3. Research articles on technology entrepreneurship from reputable journals and conferences.

Course Title: Research Project or Dissertation

Course Description:

The Research Project or Dissertation course provides students with an opportunity to engage in independent research within the field of computer science. Students will explore a specific topic, apply research methods, and produce a substantial research output. The course emphasizes critical thinking, problem-solving, and effective communication.

Learning Objectives:

1. **Research Methodology:**
   - Understand various research methods (quantitative, qualitative, or mixed methods).
   - Develop a research plan and proposal.
2. **Literature Review:**
   - Conduct a comprehensive review of existing research.
   - Identify gaps and research questions.
3. **Research Execution:**

- Collect and analyze data (if empirical research).
- Apply theoretical frameworks (if theoretical research).

Topics Covered:

1. **Research Design and Ethics:**
   - Formulating research questions.
   - Ethical considerations in research.
2. **Data Collection and Analysis:**
   - Surveys, experiments, case studies, or simulations.
   - Statistical analysis or qualitative coding.
3. **Writing and Presentation Skills:**
   - Structuring research papers.
   - Effective oral presentations.

Assessment:

- **Research Proposal:** Develop a clear and feasible research proposal.
- **Research Paper or Dissertation:** Write a well-organized research document.
- **Oral Defense:** Present research findings to peers and faculty.
- **Peer Review:** Provide constructive feedback on others' research.

Recommended Reading:

1. "Research Methods in Computer Science" by Ivan Stojmenovic.
2. Research papers from reputable conferences and journals in the student's chosen research area.

CAPITAL OPERATIONAL BUDGET FOR BACHELOR OF SCIENCE: COMPUTER SCIENCE

The programme budget is shown in the table below:

Capital Operational Budget for the Bachelor of Science: Computer Science **This budget projection calculation is based on forty students per year, covering two semesters, at a cost of R52,000 per student.**					
Description	**2023**	**2024**	**2025**	**2026**	**TOTAL**
Equity Financing	7554354	-	-	-	7554354
BSc – Computer Science	2008890	12953250	14407650	19861650	49231440
Other	152712	192708	254520	363600	963 540
TOTAL	**9715956**	**13145958**	**14662170**	**20225250**	**57749334**
Expenditure					
Administrative Costs	690190	1090800	1090800	1090800	3962589
Salaries and Wages	767876	236340	236340	236340	1476896
Equipment Costs	1524073	363600	363600	363600	2614873
Operating Costs	1088626	501768	501768	501768	2593930
Research, Training & Workshops	435451	73811	73811	73811	656883
Construction and Infrastructure	2177248	545400	545400	545400	3813448
Marketing and Studio Recording	217728	36724		36724	291175
Travelling and Accommodation	653175	491587	491587	491587	2127936
Vehicles	-	653171		363600	1016771
Other Expenses	653175	93700	93700	93700	934 274
TOTAL	**8207539**	**4086900**	**3397006**	**3797329**	**19488775**
Repayment					
Equity and Dividend Payments	1888590	1888590	1888590	1888590	7554358
Interest - 18%	339948	339948	339948	339948	1359792
Charges and Accounting	66859	66859	66859	66859	267435
TOTAL	2295396	2295396	2295396	2295396	9181584
Total Income	9715956	13145958	14662170	20225250	57749334
Total Expenditure	10502935	6382296	5692402	6092725	28670358
B/Forward	-786979	6763662	8969768	14132525	29078976
B/ Down	-786979	6763662	8969768	14132525	29078976

Join Our Celestial Journey Illuminating Minds, Igniting Innovation. Be Part of the Spark as we Unlock the Universe's Secrets, One Equation at a Time

Office of Institutional Planning and Effectiveness
The Knowledge Park I, Examination Council of Eswatini
P.O. Box D61, Ezulwini
Eswatini

www.springfieldresearch.university
frontdesk@springfieldresearch.university

+268 7619-2898
+268 2417-1634

Research Beyond Boundaries

SPRINGFIELD RESEARCH UNIVERSITY