LONDON CAPITAL COMPUTER COLLEGE

Diploma in Programming (601) – Programming Principles & Paradigms

Prerequisites: Basic programming skills or basic knowledge of computer use.	**Corequisites:** A pass or higher in Diploma in System Design or equivalence.
Aim: The course explores programming languages and paradigms, the components that comprise them, and the principles of language design, all through the analysis and comparison of a variety of languages (e.g., Pascal, C++, PROLOG, ML). This course is intended to broaden candidates' experience beyond traditional imperative programming and provide a framework for understanding what makes a programming language useful. The candidates learn about the basic components of programming languages and how they have evolved over time. This course is for candidates interested in high-level programming languages and their formal semantics. Such study enables precise reasoning about programs, their efficient implementation and easy reuse, as will be discussed in the course. The materials to be covered include operational semantics, denotational semantics, and axiomatic semantics. The course will consider imperative programming languages, functional programming languages, object-oriented programming languages, logic programming languages, and higher-level languages with sets and maps. Topics covered include type systems, abstraction mechanisms, declarativeness, and efficient implementations, concurrency and parallelism. The course objectives are: to provide an introduction to formalisms for specifying syntax and semantics of programming languages, including an introduction to the theory of formal languages; to provide an exposure to core concepts and principles in contemporary programming languages, and to explore various important programming methodologies, such as functional programming, logic programming, programming with abstract data types, and object-oriented programming; to provide a foundation in the concepts and implementation of modern programming languages by implementing imperative, functional, logic, and object-oriented programming paradigms. Upon completion of this course, candidates should be able to: understand language definition, abstractions, paradigms, basic knowledge about the major design principles; understand functional programming paradigm; understand the functional programming model – Lambda Calculus; be able to write functional programs in Lisp; understand logic programming paradigm principles – resolution and unification. be able to write Prolog programs; understand object-oriented programming principles – inheritance and dynamic binding; understand syntax definition and parsing techniques, including regular expressions, context-free grammars, parse trees and abstract syntax trees, syntax diagrams, and recursive descent parsing techniques; understand semantic definition, including attributes, binding and scoping.	
Required Materials: Student study materials	**Supplementary Materials:** Recommended textbooks and lecture notes.
Special Requirements: All topics are complicated; mostly written in abstract form, hence candidates have to read a lot outside class time.	
Intended Learning Outcomes:	**Assessment Criteria:**
1. Describe an overview of underlying principles as well as practical techniques used to design programs.	1.1 Analyse the properties of programming languages 1.2 Define programming paradigm 1.3 Describe event handling, concurrency and correctness 1.4 Explore the importance of programming 1.5 Outline programming design constraints 1.6 Distinguish compilers and interpreters
2. Describe the syntax of a programming language and demonstrate why programming language syntax is important .	2.1 Define programming language grammar 2.2 Describe extended BNF syntax 2.3 Compare programming languages syntax 2.4 Analyse how compliers and interpreters

	work
3. Demonstrate the tools for illustrating the formal semantics of programming languages.	3.1 Describe chomsky hierarchy theorem 3.2 Explore lexical analysis and parsing tools 3.3 Define syntactic analysis
4. Demonstrate how programming languages are able to use and process named variables and their contents.	4.1 Define reserved words 4.2 Outline characteristics of variables 4.3 Describe program scope 4.4 Analyse data structures for symbol tables 4.5 Describe programming reference environment 4.6 Describe dynamic scoping 4.7 Define name visibility, lifetime and overloading
5. Demonstrate how the value-level approach to programming invites the study of the space of values under the value-forming operations, and of the algebraic properties.	5.1 Describe type errors 5.2 Distinguish statically and dynamically typing 5.3 Distinguish basic and nonbasic types 5.4 Analyse subtypes, type equivalence and function types 5.5 Define polymorphism
6. Define the syntax, type system and operational semantics and demonstrate the syntax; the semantics; the type system; the features.	6.1 Be able to use expression semantics 6.2 Outline the program objects and values 6.3 Distinguish assignment statement vs assignment expression 6.4 Be able to explain flow control semantics 6.5 Describe exception handling
7. Describe functions (procedures, subroutines, methods) facilities offered by a programming language.	7.1 Define a function 7.2 Describe function parameters 7.3 Outline parameter passing mechanisms 7.4 Be able to implement recursive functions 7.5 Analyse function declarations 7.6 Distinguish function call vs function return
8. Define Memory management and demonstrate the basic memory management issues that programmers face.	8.1 Outline areas of memory 8.2 Describe arrays 8.3 Explore garbage collection 8.4 Outline an overview of the memory management techniques that are available
9. Compare and contrast functional programming with more traditional imperative (procedural) programming	9.1 Define imperative programming 9.2 Analyse procedural abstraction 9.3 Analyse imperative programming techniques
10. Demonstrate how developers use object-oriented programming techniques to implement frameworks such that the unique parts of an application can simply inherit from re-existing classes in the framework.	10.1 Analyse abstract data types 10.2 Outline characteristics of an object 10.3 Discuss OOP languages 10.4 Describe how Application framework is best with graphical user interfaces (GUIs) 10.5 Describe the design, evolution and reuse of object-oriented application

	frameworks.
11. Demonstrate how functional programming emphasises the evaluation of expressions rather than the execution of commands.	11.1 Define functional programming 11.2 Compare and contrast functional and imperative programming 11.3 Describe lambda calculus 11.4 Describe Haskell encoding
12. Demonstrate how the study of semantics also provides new ways of reasoning about the correctness of programs	12.1 Discuss the concept of Prolog programming language 12.2 Analyse the theory and practice of logic programming
13. Demonstrate how in event-driven programming the program responds to events.	13.1 Describe event-based programming paradigm 13.2 Distinguish event-driven, imperative and functional programming 13.3 Explore Graphical User Interface (GUI) applications
14. Demonstrate how in a concurrent program, several streams of operations may execute concurrently.	14.1 Outline the concurrency concepts 14.2 Describe synchronisation strategies 14.3 Explore synchronisation in Java

Recommended Learning Resources: Programming Principles & Paradigms

Text Books	• Programming Languages: Principles and Paradigms by Allen B Tucker and Robert Noonan ISBN-10: 0071254390 • Programming Language: Principles and Paradigms by Adesh Pandey ISBN-10: 1842653911 • Programming Languages: Principles and Paradigms by Maurizio Gabbrielli and Simone Martini ISBN-10: 1848829132
Study Manuals	BCE produced study packs
CD ROM	Power-point slides
Software	Lite Programming Language