Programming Languages

A programming language is a programmer's principal interface with the computer. More than just knowing how to program in a single language, programmers need to understand the different styles of programming promoted by different languages. In their professional life, they will be working with many different languages and styles at once, and will encounter many different languages over the course of their careers. Understanding the variety of programming languages and the design tradeoffs between the different programming paradigms makes it much easier to master new languages quickly. Understanding the pragmatic aspects of programming languages also requires a basic knowledge of programming language translation and runtime features such as storage allocation.

PL. Programming Languages (21 core hours)

PL/Overview [core]
PL/VirtualMachines [core]
PL/BasicLanguageTranslation [core]
PL/DeclarationsAndTypes [core]
PL/AbstractionMechanisms [core]
PL/ObjectOrientedProgramming [core])
PL/FunctionalProgramming [elective]
PL/LanguageTranslationSystems [elective]
PL/TypeSystems [elective]
PL/ProgrammingLanguageSemantics [elective]
PL/ProgrammingLanguageDesign [elective]

PL/Overview [core]

Minimum core coverage time: 2 hours

Topics:

- History of programming languages
- Brief survey of programming paradigms
- Procedural languages
- Object-oriented languages
- Functional languages
- Declarative, non-algorithmic languages
- Scripting languages
- The effects of scale on programming methodology

Learning Objectives:

1. Summarize the evolution of programming languages illustrating how this history has led to the paradigms available today.
2. Identify at least one distinguishing characteristic for each of the programming paradigms covered in this unit.
3. Evaluate the tradeoffs between the different paradigms, considering such issues as space efficiency, time efficiency (of both the computer and the programmer), safety, and power of expression.
4. Distinguish between programming-in-the-small and programming-in-the-large.

PL/VirtualMachines [core]

Minimum core coverage time: 1 hour

Topics:

- The concept of a virtual machine
- Hierarchy of virtual machines
- Intermediate languages
- Security issues arising from running code on an alien machine
-

Learning Objectives:

1. Describe the importance and power of abstraction in the context of virtual machines.

2. Explain the benefits of intermediate languages in the compilation process.
3. Evaluate the tradeoffs in performance vs. portability.
4. Explain how executable programs can breach computer system security by accessing disk files and memory.

PL/BasicLanguageTranslation [core]

Minimum core coverage time: 2 hours

Topics:

- Comparison of interpreters and compilers
- Language translation phases (lexical analysis, parsing, code generation, optimization)
- Machine-dependent and machine-independent aspects of translation

Learning Objectives:

1. Compare and contrast compiled and interpreted execution models, outlining the relative merits of each.
2. Describe the phases of program translation from source code to executable code and the files produced by these phases.
3. Explain the differences between machine-dependent and machine-independent translation and where these differences are evident in the translation process.

PL/DeclarationsAndTypes [core]

Minimum core coverage time: 3 hours

Topics:

- The conception of types as a set of values with together with a set of operations
- Declaration models (binding, visibility, scope, and lifetime)
- Overview of type-checking
- Garbage collection
-

Learning Objectives:

1. Explain the value of declaration models, especially with respect to programming-in-the-large.
2. Identify and describe the properties of a variable such as its associated address, value, scope, persistence, and size.
3. Discuss type incompatibility.
4. Demonstrate different forms of binding, visibility, scoping, and lifetime management.
5. Defend the importance of types and type-checking in providing abstraction and safety.
6. Evaluate tradeoffs in lifetime management (reference counting vs. garbage collection).

PL/AbstractionMechanisms [core]

Minimum core *coverage time: 3 hours*

Topics:

- Procedures, functions, and iterators as abstraction mechanisms
- Parameterization mechanisms (reference vs. value)
- Activation records and storage management
- Type parameters and parameterized types
- Modules in programming languages

Learning Objectives:

1. Explain how abstraction mechanisms support the creation of reusable software components.
2. Demonstrate the difference between call-by-value and call-by-reference parameter passing.
3. Defend the importance of abstractions, especially with respect to programming-in-the-large.
4. Describe how the computer system uses activation records to manage program modules and their data.

PL/ObjectOrientedProgramming [core]

Minimum core *coverage time: 10 hours*

Topics:

- Object-oriented design
- Encapsulation and information-hiding

- Separation of behavior and implementation
- Classes and subclasses
- Inheritance (overriding, dynamic dispatch)
- Polymorphism (subtype polymorphism vs. inheritance)
- Class hierarchies
- Collection classes and iteration protocols
- Internal representations of objects and method tables

Learning Objectives:
1. Justify the philosophy of object-oriented design and the concepts of encapsulation, abstraction, inheritance, and polymorphism.
2. Design, implement, test, and debug simple programs in an object-oriented programming language.
3. Describe how the class mechanism supports encapsulation and information hiding.
4. Design, implement, and test the implementation of "is-a" relationships among objects using a class hierarchy and inheritance.
5. Compare and contrast the notions of overloading and overriding methods in an object-oriented language.
6. Explain the relationship between the static structure of the class and the dynamic structure of the instances of the class.
7. Describe how iterators access the elements of a container.

PL/ FunctionalProgramming [elective]

Topics:
- Overview and motivation of functional languages
- Recursion over lists, natural numbers, trees, and other recursively-defined data
- Pragmatics (debugging by divide and conquer; persistency of data structures)
- Amortized efficiency for functional data structures
- Closures and uses of functions as data (infinite sets, streams)

Learning Objectives:
1. Outline the strengths and weaknesses of the functional programming paradigm.
2. Design, code, test, and debug programs using the functional paradigm.
3. Explain the use of functions as data, including the concept of closures.

PL/LanguageTranslationSystems [elective]

Topics:
- Application of regular expressions in lexical scanners
- Parsing (concrete and abstract syntax, abstract syntax trees)
- Application of context-free grammars in table-driven and recursive-descent parsing
- Symbol table management
- Code generation by tree walking
- Architecture-specific operations: instruction selection and register allocation
- Optimization techniques
- The use of tools in support of the translation process and the advantages thereof
- Program libraries and separate compilation
- Building syntax-directed tools

Learning Objectives:
1. Describe the steps and algorithms used by language translators.
2. Recognize the underlying formal models such as finite state automata, push-down automata and their connection to language definition through regular expressions and grammars.
3. Discuss the effectiveness of optimization.
4. Explain the impact of a separate compilation facility and the existence of program libraries on the compilation process.

PL/TypeSystems [elective]

Topics:

- Data type as set of values with set of operations
- Data types
- Elementary types
- Product and coproduct types
- Algebraic types
- Recursive types
- Arrow (function) types
- Parameterized types
- Type-checking models
- Semantic models of user-defined types
- Type abbreviations
- Abstract data types
- Type equality
- Parametric polymorphism
- Subtype polymorphism
- Type-checking algorithms

Learning Objectives:
1. Formalize the notion of typing.
2. Describe each of the elementary data types.
3. Explain the concept of an abstract data type.
4. Recognize the importance of typing for abstraction and safety.
5. Differentiate between static and dynamic typing.
6. Differentiate between type declarations and type inference.
7. Evaluate languages with regard to typing.

PL/ProgrammingLanguageSemantics [elective]

Topics:
- Informal semantics
- Overview of formal semantics
- Denotational semantics
- Axiomatic semantics
- Operational semantics

Learning Objectives:
1. Explain the importance of formal semantics.
2. Differentiate between formal and informal semantics.
3. Describe the different approaches to formal semantics.
4. Evaluate the different approaches to formal semantics.

PL/ProgrammingLanguageDesign [elective]

Topics:
- General principles of language design
- Design goals
- Typing regimes
- Data structure models
- Control structure models
- Abstraction mechanisms

Learning Objectives:
1. Evaluate the impact of different typing regimes on language design, language usage, and the translation process.
2. Explain the role of different abstraction mechanisms in the creation of user-defined facilities