

Haskell Programming

Audience Haskell Programming

The course Haskell Programming course is intended for anyone who wants to learn programming in the functional programming language Haskell.

Prerequisites Course Haskell Programming

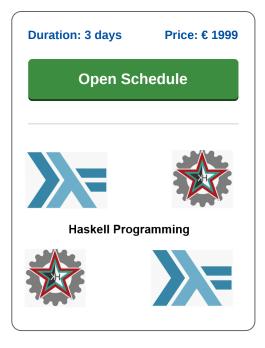
In order to participate in this course basic knowledge of programming in another programming language is beneficial to understanding, but is not required.

Realization Training Haskell Programming

The theory is discussed on the basis of presentation slides. The theory is further explained through demos. After discussing a module, there is the possibility to practice. Course times are from 9.30 to 16.30.

Course Certification Haskell Programming

After successful completion of the course, the participants receive an official certificate Haskell Programming.



Content Course Haskell Programming

In the course Haskell Programming participants learn to program in the pure functional programming language Haskell. Unlike conventional languages such as C and Java, functional languages perform calculations by combining separate mathematical functions. A functional language like Haskell also naturally has support for multithreading and parallelism, which results in good performance.

The course starts with an introduction to the basic syntax of Haskell and the main characteristics of the language such as laziness in expression evaluation with a thunk data structure, static typing, modularity and dealing with multiple threads.

Data Types and Operators

Next attention is paid to the data types and operators of the language. Data structures such as lists, tuples and records are also discussed. And exceptions and accessing the file system are also treated.

Functions

Part of the program of the course are the definition and declaration of functions in Haskell with argument lists, pattern matching and quards. Recursive functions, higher order functions and lambda expressions are discussed as well.

Modules

Then the various Haskell modules such as the Prelude, List, Char, Set and Map module are reviewed. And it is explained how user defined modules and packages are created. This also includes the discussion of qualified imports and the global namespace.

Haskell Containers

Finally attention is paid to Haskell containers such as classes, objects, Trees, Binary Trees, Functors and Foldables and the characteristic functional programming concept of monads.



Modules Course Haskell Programming

| Module 1 : Haskell Intro | Module 2 : Types and Operators | Module 3 : Functions |
|---|---|--|
| What is Haskell? | Numbers and Characters | Function Declaration |
| Functional Programming | Strings and Booleans | Function Names |
| Evaluation Engine | Lists | Argument Lists |
| Expression Evaluation | List Comprehensions | Function Definition |
| Haskells Laziness | Tuples | Pattern Matching |
| Thunk Data Structure | Haskell Operators | Guards |
| Haskells Modularity | Sequence Operator | Where Clause |
| Handling Multiple Threads | Control Flow | Recursion |
| Static Typing | Exceptions | Lambda Expressions |
| Installing Haskell | Type Class | Higher Order Functions |
| Haskell Compiler | Records | Head and Tail |
| Stack Installer | Default Values Idiom | Null and Init |
| Haskell Platform | Files and Streams | Reverse and Take |
| Module 4 : Modules | Module 5 : Containers | Module 6 : Monads |
| Packages and Modules | Classes and Instances | Applicative Functors |
| import Statement | Maps | Monadic Rules |
| | | Worldale Males |
| Prelude Module | Sets | Left Identity Law |
| Prelude Module Global Namespace | Sets Trees | |
| | | Left Identity Law |
| Global Namespace | Trees | Left Identity Law Right Identity Law |
| Global Namespace Qualified Imports | Trees Binary Trees | Left Identity Law Right Identity Law Associativity |
| Global Namespace Qualified Imports List Module | Trees Binary Trees Graphs | Left Identity Law Right Identity Law Associativity Combinators for State |
| Global Namespace Qualified Imports List Module Char Module | Trees Binary Trees Graphs Type Classes | Left Identity Law Right Identity Law Associativity Combinators for State Dissecting Combinators |
| Global Namespace Qualified Imports List Module Char Module Map Module | Trees Binary Trees Graphs Type Classes Built-in Type Classes | Left Identity Law Right Identity Law Associativity Combinators for State Dissecting Combinators do Notation |
| Global Namespace Qualified Imports List Module Char Module Map Module Set Module | Trees Binary Trees Graphs Type Classes Built-in Type Classes Polymorphism | Left Identity Law Right Identity Law Associativity Combinators for State Dissecting Combinators do Notation State and Lenses |