100 Scala interview questions to hire the best developers

Questions

1. Can you explain what Scala is and why it's used?

2. What are the main differences between val and var in Scala?

3. Explain what an immutable variable is. How is it useful?

4. What is a function in Scala, and how do you define one?

5. What is the difference between a function and a method in Scala?

6. How do you create a class in Scala?

7. What is an object in Scala, and how does it differ from a class?

8. Explain what a case class is and when you might use it.

9. What is a trait in Scala, and how is it different from a class?

10. How do you implement inheritance in Scala?

11. What is the purpose of the 'main' method in Scala?

12. Can you describe what pattern matching is in Scala?

13. What is a Scala list, and how do you add elements to it?

14. How do you iterate through a list in Scala?

15. What are the advantages of using immutable collections in Scala?

16. What is an Option in Scala, and why is it useful?

17. Explain the purpose of the 'for' comprehension in Scala.

18. What is a tuple in Scala, and how do you access its elements?

19. Can you describe what a higher-order function is in Scala?

20. How do you handle exceptions in Scala?

21. What is the REPL in Scala, and how is it used?

22. How do you compile and run a Scala program?

23. Explain the concept of currying in Scala.

24. What is the difference between 'apply' and 'unapply' methods in Scala?

25. How do you define default parameter values in a Scala function?

26. Can you explain the concept of type inference in Scala?

27. What are some of the benefits of using Scala over Java?

28. What is the difference between val and var in Scala? Can you give simple examples?

29. Explain what an Option in Scala is, and why we use it.

30. What is a Scala Trait, and how is it different from a Class?

31. How do you define a function in Scala? Show a simple example.

32. What does immutable mean in the context of Scala collections?

33. Explain what pattern matching is in Scala and show a basic example.

34. What is the purpose of the Unit type in Scala?

35. How do you create a simple for loop in Scala?

36. What is a Scala case class, and what are some of its benefits?

37. Describe what you understand about Scala's type inference.

38. What is the difference between List and Array in Scala?

39. How can you define default parameter values for a function in Scala?

40. Explain what you know about string interpolation in Scala.

41. What does it mean for a function to be a 'first-class citizen' in Scala?

42. Describe a simple use case for using if/else statements in Scala.

43. How would you create a simple class with a constructor in Scala?

44. What is the role of the main method in a Scala program?

45. Can you explain what a higher-order function is in Scala?

46. What are some advantages of using Scala over Java, in your opinion?

47. Explain what you know about the concept of immutability and how Scala supports it.

48. What are the main differences between mutable and immutable collections? When would you choose one over the other?

49. Can you describe how to handle exceptions in Scala using try-catch blocks?

50. How do you define a method within a class in Scala?

51. What are the basic building blocks of a simple Scala application?

52. How can you add extra behavior (methods) to a class after it's been defined in Scala, and what is this concept called?

53. Explain the concept of Type Classes in Scala and provide a practical example of how they are used for ad-hoc polymorphism.

54. How does the Scala compiler resolve implicit parameters? Describe a scenario where implicit resolution might fail and how to troubleshoot it.

55. Discuss the differences between CanBuildFrom, Builder, and immutable data structures when constructing collections in Scala. When would you prefer using one over the other?

56. Describe how you can use the ExecutionContext in Scala's Future to manage concurrency. Explain the potential pitfalls of using a global ExecutionContext and how to avoid them.

57. Explain the concept of Monads in Scala. How do they help in managing side effects and sequencing operations? Provide a simple example.

58. What are variance annotations (+/-) in Scala? How do they affect type safety, especially when working with generic collections?

59. Explain the use of type aliases and structural types in Scala. Provide examples of when these features can improve code readability and flexibility.

60. Discuss the advantages and disadvantages of using Akka Actors for building concurrent systems in Scala. How do you handle actor failures and ensure fault tolerance?

61. How does Scala's for comprehension desugar into flatMap, map, and filter calls? Explain scenarios where using for comprehension enhances code clarity.

62. Explain the concept of a Monoid in Scala, and give a real-world example (other than the usual string concatenation or numeric addition) of how it can be useful.

63. How do you implement custom extractors in Scala using the unapply method? What are some use cases for custom extractors beyond pattern matching?

64. Describe how you would use Scala's reflection capabilities to dynamically inspect and manipulate classes at runtime. What are the performance implications of using reflection?

65. Explain how you would implement a custom DSL (Domain Specific Language) in Scala using a combination of implicit conversions, operators, and other language features.

66. Discuss techniques for optimizing Scala code for performance, including minimizing object allocations, using specialized collections, and leveraging tail recursion.

67. How can you achieve compile-time metaprogramming in Scala using macros or Scala 3's new features? What are the advantages and disadvantages compared to runtime reflection?

68. Explain how to use tagged types in Scala to add compile-time checking to your code and prevent logical errors. Provide an example of how tagged types can improve type safety.

69. Describe the use of phantom types in Scala. Provide a scenario where phantom types enhance type safety beyond what regular types can provide.

70. Discuss strategies for handling errors and exceptions in asynchronous Scala code (e.g., using Futures or Akka). How do you ensure proper error propagation and recovery?

71. Explain the concept of lenses in Scala and their use in functional data manipulation. How can lenses simplify working with deeply nested immutable data structures?

72. Describe techniques for testing asynchronous Scala code effectively. What are some common pitfalls in testing asynchronous code and how can you avoid them?

73. How does Scala's PartialFunction differ from a regular function? Give an example of where PartialFunction is particularly useful, such as in actor message handling or routing.

74. How does Scala's type system handle variance, and why is this important in generic programming?

75. Explain the concept of a monad in Scala, providing a practical example beyond the typical Option or List monads.

76. Describe the Actor Model and its implementation in Scala using Akka, including strategies for handling actor failures.

77. How does Scala's implicit resolution mechanism work, and what are some potential pitfalls to avoid when using implicits?

78. What are some advanced techniques for optimizing Scala code for performance, such as avoiding boxing or using specialized collections?

79. Explain the concept of a type class in Scala, and how it differs from traditional object-oriented interfaces.

80. Describe how you would implement a custom collection in Scala, including considerations for performance and memory usage.

81. How does Scala's CanBuildFrom trait work, and why is it important for collection transformations?

82. Explain how you can use Scala's macros to generate code at compile time, and what are the benefits and drawbacks of this approach?

83. Describe different concurrency models available in Scala, comparing and contrasting their use cases and trade-offs.

84. How do you handle asynchronous programming in Scala using Futures and Promises, including error handling and composition?

85. Explain advanced type system features like path-dependent types, structural types and dependent method types with examples.

86. Describe different serialization techniques available in scala, emphasizing considerations for versioning and migration of data.

87. How do you approach debugging and profiling Scala applications, highlighting any specific tools or techniques?

88. Explain how Scala's view bounds and context bounds work, and what situations they are best suited for.

89. How would you design a fault-tolerant distributed system using Akka, including strategies for data consistency and recovery?

90. Describe best practices for testing Scala code, including the use of mocking frameworks and property-based testing.

91. What are some advanced techniques for working with Scala's collections API, such as using views or custom combinators?

92. Explain how Scala's dynamic method invocation works, and what use cases it is appropriate for.

93. Describe the concept of a Monad Transformer and provide an example of how it solves a specific problem.

94. Explain how you would implement a custom DSL (Domain Specific Language) in Scala, and what benefits it provides.

95. Describe how Scala Native works, and what are the advantages and disadvantages of using it compared to the JVM?

96. How does Scala.js work and what are the implications when writing front end code in scala?

97. Explain how you would use Shapeless to perform generic programming tasks in Scala, such as deriving type class instances.

98. Describe what the Cake Pattern is, and when and why you might use it for dependency injection in Scala.

99. Discuss the trade-offs between using immutable data structures and mutable data structures in concurrent Scala programs.