1. What is Haskell?

a) A functional programming language
b) An imperative programming language
c) A procedural programming language
d) An object-oriented programming language

Answer: a) A functional programming language

Explanation: Haskell is a pure functional programming language that emphasizes the use of functions to solve problems. It is based on lambda calculus and is known for its strong type system, lazy evaluation, and purity.

2. What is lazy evaluation in Haskell?

a) Functions are only executed when they are needed
b) Functions are executed immediately
c) Functions are never executed
d) Functions are executed randomly

Answer: a) Functions are only executed when they are needed

Explanation: Haskell uses lazy evaluation, which means that expressions are not evaluated until they are needed. This can save time and memory, since Haskell only evaluates what is needed to produce the final result.

3. What is a higher-order function in Haskell?

a) A function that takes another function as an argument
b) A function that returns another function as a result

c) A function that can do both of the above
d) A function that does not take any arguments

Answer: c) A function that can take another function as an argument and/or return another function as a result

Explanation: Haskell allows functions to be passed as arguments to other functions and returned as results, just like any other value. This makes it possible to write more abstract and reusable code.

4. Which of the following is a correct syntax for defining a function in Haskell?

a) functionName = parameter1 parameter2 = expression
b) functionName parameter1 parameter2 = expression
c) parameter1 parameter2 functionName = expression
d) parameter1 parameter2 = functionName expression

Explanation: In Haskell, functions are defined using the syntax functionName parameter1 parameter2 = expression, where functionName is the name of the function, parameter1, and parameter2 are the arguments, and expression is the body of the function.

5. What is a type signature in Haskell?

a) A way to specify the type of a function or value
b) A way to specify the name of a function or value
c) A way to specify the value of a function or value
d) A way to specify the scope of a function or value

Answer: a) A way to specify the type of a function or value

Explanation: Haskell is a statically typed language, which means that types are checked at compile time. A type signature is a way to specify the type of a function or value, which helps the compiler catch type errors before the code is run.

6. What is pattern matching in Haskell?

a) A way to match patterns of data to specific expressions
b) A way to match patterns of expressions to specific data
c) A way to match patterns of functions to specific expressions
d) A way to match patterns of expressions to specific functions

Answer: a) A way to match patterns of data to specific expressions

Explanation: Haskell allows you to use pattern matching to match patterns of data to specific expressions. This is useful when dealing with data structures like lists or tuples, and can simplify code by allowing you to handle different cases in a single function.

7. What is a lambda function in Haskell?

a) A function that takes no arguments
b) A function that returns no value
c) A function that is defined inline
d) A function that is defined externally

Answer: c) A function that is defined inline

Explanation: A lambda function, also known as an anonymous function, is a function that is defined inline, without a name. Lambda functions are useful for writing short, one-off functions that are only used once.

8. What is currying in Haskell?

a) A technique for transforming a function that takes multiple arguments into a series of functions that each take a single argument
b) A technique for transforming a function that takes a single argument into a function that takes multiple arguments
c) A technique for transforming a function that takes no arguments into a function that takes one or more arguments
d) A technique for transforming a function that returns a value into a function that does not return a value

Answer: a) A technique for transforming a function that takes multiple arguments into a series of functions that each take a single argument

Explanation: In Haskell, currying is a technique for transforming a function that takes multiple arguments into a series of functions that each take a single argument. This can make it easier to compose functions and create more reusable code.

9. What is function composition in Haskell?

a) Combining two or more functions to create a new function
b) Breaking a function down into smaller functions
c) Creating a function that takes a function as an argument
d) Creating a function that returns a function as a result

Answer: a) Combining two or more functions to create a new function

Explanation: Function composition is the process of combining two or more functions to create a new function. In Haskell, the composition operator (.) can be used to create a new function that applies one function to the output of another function.

10. What is a monad in Haskell?

a) A type of data structure
b) A design pattern for handling side effects
c) A functional programming paradigm
d) A way to define functions in terms of recursion

Answer: b) A design pattern for handling side effects

Explanation: In Haskell, a monad is a design pattern for handling side effects, such as input/output or state. Monads provide a way to sequence computations that involve side effects, while still maintaining the purity of the functional programming paradigm.

11. What is a typeclass in Haskell?

a) A way to define a new type of data
b) A way to define a set of operations that can be performed on a type of data
c) A way to define a set of functions that operate on a type of data
d) A way to define a set of constraints on a type of data

Answer: b) A way to define a set of operations that can be performed on a type of data

Explanation: In Haskell, a typeclass is a way to define a set of operations that can be performed on a type of data. Typeclasses are similar to interfaces in object-oriented programming, and allow different types of data to be used interchangeably in functions that operate on them.

12. What is the difference between the Maybe and Either types in Haskell?

a) The Maybe type can have only one possible value, while the Either type can have multiple possible values
b) The Maybe type represents a value that may or may not be present, while the Either type represents a value that can have one of two possible states
c) The Maybe type represents a value that can have one of two possible states, while the Either type represents a value that may or may not be present
d) The Maybe and Either types are the same and can be used interchangeably

Answer: b) The Maybe type represents a value that may or may not be present, while the Either type represents a value that can have one of two possible states

Explanation: The Maybe type is used to represent a value that may or may not be present, while the Either type is used to represent a value that can have one of two possible states. For example, the Either type can be used to represent the result of a computation, where the result can be either an error or a successful value.

13. Which of the following is not a built-in data type in Haskell?

a) Integer
b) Char
c) Boolean
d) Double

Answer: c) Boolean

Explanation: Boolean is not a built-in data type in Haskell, but it can be easily defined using the following type synonym: type Boolean = Bool.

14. Which of the following functions is used to apply a function to each element of a list in Haskell?

a) map
b) filter
c) foldr
d) foldl

Answer: a) map

Explanation: The map function in Haskell is used to apply a function to each element of a list, producing a new list with the results.

15. What is the difference between a function and a lambda function in Haskell?

a) A lambda function is a type of function that is defined inline and does not have a name, while a function is a named block of code that can be reused

b) A function is a type of lambda function that is defined using the “fun” keyword, while a lambda function is defined using the “lambda” keyword
c) A function and a lambda function are the same thing
d) A lambda function is a type of function that can be used only in certain contexts, while a function can be used anywhere

Answer: a) A lambda function is a type of function that is defined inline and does not have a name, while a function is a named block of code that can be reused

Explanation: In Haskell, a lambda function is a type of function that is defined inline and does not have a name, while a function is a named block of code that can be reused. Lambda functions can be useful for creating small, one-time-use functions.

16. What is recursion in Haskell?

a) A technique for defining a function in terms of itself
b) A technique for generating random values
c) A technique for transforming one type of data into another type of data
d) A technique for optimizing the performance of functions

Answer: a) A technique for defining a function in terms of itself

Explanation: In Haskell, recursion is a technique for defining a function in terms of itself. This can be used to create functions that operate on recursive data structures, or to create functions that have a repetitive pattern of behavior.

17. What is a list comprehension in Haskell?

a) A way to generate a list based on a set of input values and one or more output expressions
b) A way to generate a random list of values
c) A way to transform one type of data into another type of data
d) A way to optimize the performance of functions

Answer: a) A way to generate a list based on a set of input values and one or more output expressions

Explanation: In Haskell, a list comprehension is a way to generate a list based on a set of input values and one or more output expressions. This can be used to create more concise and expressive code, and is a fundamental concept in Haskell programming.

18. What is a functor in Haskell?

a) A typeclass that defines a way to apply a function to the contents of a data structure
b) A typeclass that defines a way to transform one type of data into another type of data
c) A typeclass that defines a way to generate random values
d) A typeclass that defines a way to optimize the performance of functions

Answer: a) A typeclass that defines a way to apply a function to the contents of a data structure

Explanation: In Haskell, a functor is a typeclass that defines a way to apply a function to the contents of a data structure. This allows for more concise and expressive code, and is a fundamental concept in Haskell programming.

19. What is a monoid in Haskell?

a) A typeclass that defines a way to combine two values of the same type
b) A typeclass that defines a way to combine two values of different types
c) A typeclass that defines a way to compare two values of the same type
d) A typeclass that defines a way to compare two values of different types

Answer: a) A typeclass that defines a way to combine two values of the same type

Explanation: In Haskell, a monoid is a typeclass that defines a way to combine two values of the same type. This allows for more modular and reusable code, and is a fundamental concept in Haskell programming.

20. What is a fold in Haskell?

a) A higher-order function that applies a function to each element of a list
b) A way to define a recursive function in Haskell
c) A way to generate a list based on a set of input values and one or more output expressions
d) A way to specify the type of a variable or function in Haskell

Answer: a) A higher-order function that applies a function to each element of a list

Explanation: In Haskell, a fold is a higher-order function that applies a function to each element of a list, accumulating a value along the way. This can be used to solve many problems in a concise and efficient way.

21. What is laziness in Haskell?

a) A way to optimize the performance of functions
b) A way to delay the evaluation of expressions until they are actually needed
c) A way to generate random values
d) A way to transform one type of data into another type of data

Answer: b) A way to delay the evaluation of expressions until they are actually needed

Explanation: In Haskell, laziness is a way to delay the evaluation of expressions until they are actually needed. This can be used to create more efficient and expressive code, and is a fundamental concept in Haskell programming.

22. What is a partial function in Haskell?

a) A function that is only defined for some values of its input
b) A function that is defined for all values of its input
c) A function that generates random values
d) A function that transforms one type of data into another

Answer: a) A function that is only defined for some values of its input

Explanation: In Haskell, a partial function is a function that is only defined for some values of its input. This can lead to unexpected behavior or errors at runtime, and should be avoided whenever possible.

23. What is a module in Haskell?

a) A collection of functions and types that can be imported and used in other programs
b) A way to define a recursive function in Haskell
c) A way to generate a list based on a set of input values and one or more output expressions
d) A way to specify the type of a variable or function in Haskell

Answer: a) A collection of functions and types that can be imported and used in other programs

Explanation: In Haskell, a module is a collection of functions and types that can be imported and used in other programs. This can be used to create more modular and reusable code, and is a fundamental concept in Haskell programming.

24. What is a guard in Haskell?

a) A way to define a function that takes multiple arguments as a sequence of functions that each take one argument
b) A way to specify the type of a variable or function in Haskell
c) A way to generate a list based on a set of input values and one or more output expressions
d) A way to add conditional expressions to a function definition

Answer: d) A way to add conditional expressions to a function definition

Explanation: In Haskell, a guard is a way to add conditional expressions to a function definition. This can be used to create more complex and flexible functions that can handle different inputs and outputs depending on specific conditions.

25. What is the difference between a list and a tuple in Haskell?

a) A list is mutable and a tuple is immutable
b) A list is a sequence of elements of the same type, while a tuple can contain elements of different types
c) A list can have a variable length, while a tuple has a fixed length
d) A list can be used for pattern matching, while a tuple cannot

Answer: b) A list is a sequence of elements of the same type, while a tuple can contain elements of different types

Explanation: In Haskell, a list is a sequence of elements of the same type, while a tuple can contain elements of different types. Additionally, a list can have a variable length, while a tuple has a fixed length.

26. What is the difference between a function and a procedure in Haskell?

a) A function returns a value, while a procedure does not
b) A function can have side effects, while a procedure cannot
c) A function takes input parameters, while a procedure does not
d) There is no difference between a function and a procedure in Haskell

Answer: a) A function returns a value, while a procedure does not

Explanation: In Haskell, a function is a piece of code that takes input parameters, performs some computation, and returns a value. A procedure, on the other hand, is a piece of code that performs some computation but does not return a value.

27. What is a type class in Haskell?

a) A set of types that share common properties and operations
b) A way to represent computations as a sequence of steps that can be composed and combined
c) A way to ensure type safety and to catch errors at compile time
d) A way to define a set of functions and operations that can be used on a particular type or set of types

Answer: d) A way to define a set of functions and operations that can be used on a particular type or set of types

Explanation: In Haskell, a type class is a way to define a set of functions and operations that can be used on a particular type or set of types. This allows for more modular and reusable code, and is a fundamental concept in Haskell programming.

28. Which of the following is NOT a fundamental data type in Haskell?

a) Integer
b) Double
c) Char
d) Object

Answer: d) Object

Explanation: Object is not a fundamental data type in Haskell. The fundamental data types in Haskell are Integer, Double, Char, Bool, and the various list and tuple types.

29. Which of the following is NOT a list operation in Haskell?

a) map
b) filter
c) reduce
d) fold

Answer: c) reduce

Explanation: reduce is not a list operation in Haskell. The list operations in Haskell are map, filter, fold, and various others.

30. Which of the following is the correct syntax for a list comprehension in Haskell?

a) [x | x <- [1,2,3]]
b) [x in [1,2,3] | x]
c) [x | x = [1,2,3]]
d) [x | [1,2,3] -> x]

Answer: a) [x | x <- [1,2,3]]

Explanation: The correct syntax for a list comprehension in Haskell is [output expression | generator expression(s)]. In this case, the generator expression is x <- [1,2,3], and the output expression is simply x.

31. Which of the following is the correct way to define a recursive function that calculates the factorial of a number in Haskell?

a) factorial n = if n == 0 then 1 else n * factorial n-1
b) factorial n = if n == 0 then 1 else n * factorial (n-1)
c) factorial n = if n == 1 then 1 else n * factorial n-1
d) factorial n = if n == 1 then 1 else n * factorial (n-1)

Answer: b) factorial n = if n == 0 then 1 else n * factorial (n-1)

Explanation: The correct way to define a recursive function that calculates the factorial of a number in Haskell is to use the if-then-else

32. Which of the following is a type of Haskell function that takes a function as an argument and returns a new function?

a) Higher-order function
b) Recursive function
c) Lambda function
d) Anonymous function

Answer: a) Higher-order function

Explanation: A higher-order function is a type of Haskell function that takes a function as an argument and returns a new function. Higher-order functions are a fundamental concept in functional programming, and are used extensively in Haskell.

33. Which of the following is the correct syntax for defining a data type in Haskell?

a) data Type = Constructor
b) type Constructor = Type

c) Constructor Type = data
d) data Type Constructor

Answer: a) data Type = Constructor

Explanation: The correct syntax for defining a data type in Haskell is data Type = Constructor. This creates a new data type called Type, with a single constructor called Constructor.

34. Which of the following is NOT a standard Haskell module?

a) Data.List
b) System.IO
c) Control.Monad
d) Math.Matrix

Answer: d) Math.Matrix

Explanation: Math.Matrix is not a standard Haskell module. The standard Haskell modules include Data.List, System.IO, Control.Monad, and many others.

35. What is the purpose of the do notation in Haskell?

a) To define anonymous functions
b) To define recursive functions
c) To define monadic computations in a more readable way
d) To define list comprehensions

Answer: c) To define monadic computations in a more readable way

Explanation: The do notation in Haskell is used to define monadic computations in a more readable way. Monads can be difficult to read and understand when written in a purely functional style, and the do notation provides a more imperative-looking syntax that can make the code easier to read.

36. What is the purpose of the guard syntax in Haskell?

a) To test for a condition and execute code based on the result
b) To define anonymous functions
c) To define recursive functions
d) To define list comprehensions

Answer: a) To test for a condition and execute code based on the result

Explanation: The guard syntax in Haskell is used to test for a condition and execute code based on the result. Guards are typically used in function definitions to handle different cases based on the inputs, and can be a powerful tool for creating more flexible and modular code.

37. Which of the following is NOT a built-in type class in Haskell?

a) Eq
b) Ord
c) Num
d) Str

Answer: d) Str

Explanation: Str is not a built-in type class in Haskell. The built-in type classes in Haskell include Eq, Ord, Num, and many others.

38. Which of the following is the correct way to define a recursive function in Haskell?

a) f x = x + 1
b) f x = if x == 0 then 1 else x * f (x-1)
c) f x y = x + y
d) f x = map (*2) x

Answer: b) f x = if x == 0 then 1 else x * f (x-1)

Explanation: The correct way to define a recursive function in Haskell is to use an if-else statement or a pattern matching to define the base case and a recursive call to define the general case. Option (b) is an example of such a function that calculates the factorial of a number.

39. Which of the following is the correct way to define a lambda function in Haskell?

a) (\x y -> x + y)
b) \x y -> x + y
c) \x -> map (*2) x
d) (\x -> x + 1) y

Answer: b) \x y -> x + y

Explanation: The correct way to define a lambda function in Haskell is to use the backslash symbol followed by the arguments, separated by spaces or arrows, and then the expression to be evaluated. Option (b) is an example of such a function that takes two arguments and returns their sum.

40. Which of the following is the correct way to define a list comprehension in Haskell?

a) [x + y | x <- [1,2,3], y <- [4,5,6]]
b) [x + y | x + y <- [1,2,3,4,5,6]]
c) [x + y | x <- [1,2,3], y = [4,5,6]]
d) [x * y | x <- [1,2,3], y <- [4,5,6], x + y == 8]

Answer: a) [x + y | x <- [1,2,3], y <- [4,5,6]]

Explanation: The correct way to define a list comprehension in Haskell is to use square brackets and specify the expression to be evaluated, followed by a pipe symbol, and then one or more generators that define the values to be used. Option (a) is an example of such a list comprehension that calculates the sum of pairs of numbers taken from two lists.

41. Which of the following is NOT a type of Haskell function composition operator?

a) .
b) $
c) <>
d) >>=

Answer: c) <>

Explanation: <> is not a type of Haskell function composition operator. The function composition operators in Haskell include the dot operator (.), the dollar sign operator ($), and the bind operator (>>=), among others.

42. Which of the following is the correct way to define a type synonym in Haskell?

a) type MyType = [Int]
b) type [MyType] = Int
c) type MyType Int = [Int]
d) type MyType = [a]

Answer: a) type MyType = [Int]

Explanation: The correct way to define a type synonym in Haskell is to use the type keyword followed by the name of the new type and its definition. Option (a) is an example of such a definition that creates a new type called MyType, which is a list of integers.

43. Which of the following is the correct way to define a type class in Haskell?

a) class MyType where
b) class (Eq a) => MyType a where
c) class MyType a :: Eq where
d) class (MyType a) => Eq a where

Answer: b) class (Eq a) => MyType a where

Explanation: The correct way to define a type class in Haskell is to use the class keyword followed by the name of the new class and its type parameter(s), followed by any constraints on the type parameter(s) and the class functions. Option (b) is an example of such a definition that creates a new type class called MyType with a type parameter a that is constrained by the Eq class and with its own set of functions.

44. Which of the following is the correct way to define an instance of a type class in Haskell?

a) instance MyType where
b) instance Eq a => MyType a where
c) instance MyType a :: Eq where
d) instance Eq a => MyType a :: Eq where

Answer: b) instance Eq a => MyType a where

Explanation: The correct way to define an instance of a type class in Haskell is to use the instance keyword followed by the name of the type class and the type parameter(s) for the instance, followed by any constraints on the type parameter(s) and the implementation of the class functions. Option (b) is an example of such a definition that defines an instance of the MyType class for a type parameter a that is constrained by the Eq class and with its own implementation of the class functions.

45. Which of the following is the correct way to import a module in Haskell?

a) import "MyModule"
b) import MyModule

c) include MyModule
d) use MyModule

Answer: b) import MyModule

Explanation: The correct way to import a module in Haskell is to use the import keyword followed by the name of the module to be imported. Option (b) is an example of such an import statement that imports the module called MyModule.

46. Which of the following is the correct way to define a function with a default parameter value in Haskell?

a) f x = x + 1 (default 0)
b) f x y = x + y (default 1)
c) f x y z = x + y + z (default 0)
d) Haskell does not support default parameter values

Answer: d) Haskell does not support default parameter values

Explanation: Haskell does not support default parameter values for functions. However, it is possible to achieve similar functionality using optional arguments or by defining multiple versions of the same function with different numbers of arguments.

47. What is the output of the following code?

```kotlin
f :: Int -> Int -> Int
f x y = x + y
```

```
g :: Int -> Int
g = f 2
main = print (g 3)
```

a) 5
b) 6
c) Error: Too few arguments to function ‘g’
d) Error: No instance for (Show (Int -> Int)) arising from a use of ‘print’

Answer: a) 5

Explanation: The code defines two functions f and g. Function f takes two integers and returns their sum. Function g takes a single integer and returns the result of calling function f with 2 as its first argument and the argument to g as its second argument. The main function calls g with an argument of 3 and prints the result. Therefore, the output will be 5.

48. What does the where keyword do in Haskell?

a) It declares a function that is local to another function.
b) It declares a type synonym.
c) It declares a new data type.
d) It declares a new type class.

Answer: a) It declares a function that is local to another function.

Explanation: In Haskell, the where keyword is used to declare a function that is local to another function. The local function can be used within the body of the outer function, but it is not visible outside of the outer function.

This is useful for breaking a larger function into smaller pieces, or for defining helper functions that are only used within a specific context.

49. What is the output of the following code?

```
python
main = print (take 10 (cycle “abc”))
```

a) “abcabcabca”
b) [“abc”, “abc”, “abc”, “abc”, “abc”, “abc”, “abc”, “abc”, “abc”, “abc”]
c) [“a”, “b”, “c”, “a”, “b”, “c”, “a”, “b”, “c”, “a”]
d) [“a”, “b”, “c”, “a”, “b”, “c”, “a”, “b”, “c”, “b”]

Answer: a) “abcabcabca”

Explanation: The ‘cycle’ function creates an infinite list by repeating its argument. In this case, the argument is the string “abc”. The ‘take’ function is then used to take the first 10 elements of the infinite list. The result is a string with 10 characters, which is “abcabcabca”.