3.

3.1. NIL

3.2. (/ (+ 8 12) 2)

3.3. (+ (* 3 3) (* 4 4))

3.7. (defun MILES-PER-GALLON (INITIAL-ODOMETER-READING FINAL-ODOMETER-READING GALLONS-CONSUMED)(/ (- FINAL-ODOMETER-READING INITIAL-ODOMETER-READING) GALLONS-CONSUMED))

3.10.

3.10.1. Missing quote on the list. Without a quote, Lisp treats the as a function and quick, brown, and fox as its arguments. The intended meaning is to use a literal list of symbols. (third '(the quick brown fox))

3.10.2. Unquoted symbols causing evaluation errors. The words and and is are intended to be literal symbols in the list, but without quotes Lisp attempts to evaluate them as variables or functions. (list 2 'and 2 'is 4)

3.10.3. Incorrect quote causing an unintended literal rather than an evaluated expression. Quoting the entire (length (list t t t t)) prevents it from being evaluated, so Lisp ends up trying to add a number to a list. (+ 1 (length (list t t t t)))

3.10.4. Missing quote on the list meant to be literal. While 'patrick is correctly quoted, (seymour marvin) is interpreted as a function call with seymour as the function and marvin as its argument. To treat it as a literal list, it must be quoted. (cons 'patrick '(seymour marvin))

3.10.5. Unquoted symbols within the list causing evaluation errors. In the expression (list seymour marvin), the symbols seymour and marvin will be treated as variables (or function calls) unless they are quoted. (cons 'patrick (list 'seymour 'marvin))

3.20. When you evaluate (mystery '(dancing bear)), the function swaps the two elements in the list, returning (bear dancing). However, evaluating (mystery 'dancing 'bear) causes an error because the function is defined to take a single argument while two arguments are provided. Evaluating (mystery '(zowie)) results in (nil zowie) since the list contains only one element—there is no second element, so it defaults to NIL—and (mystery (list 'first 'second)) swaps the two symbols in the list (first second), yielding (second first).

3.21. For the first definition, the error is that both parameters are quoted in the body—using quotes on x and y prevents their values from being used, so instead of inserting the passed-in values, the function always returns the literal symbols; the fix is to quote only the literal words, as in (list 'all x 'is y). In the second definition, the error is twofold: the parameter list contains only one variable (x) yet the body refers to y (and even includes a spurious (y) expression that attempts to call y as a function), so y is unbound; the correction is to include y in the parameter list and remove the errant (y), yielding (defun speak (x y) (list 'all x 'is y)). The third definition errs by specifying its parameters as ((x) (y)) rather than as simple symbols (i.e. (x y)), and in the function body, some literal words like all

and is are not quoted, causing them to be treated as variables rather than constants; to fix this, write the parameter list as (x y) and quote the literal symbols, for example (defun speak (x y) (list 'all x 'is y)).

3.25. (list 'cons t nil)
This expression simply constructs a list whose first element is the symbol CONS, the second is T, and the third is NIL. As a result, it evaluates to a literal list that prints as (cons t nil).
(eval (list 'cons t nil))
Here the inner call builds the list (cons t nil), which is then evaluated as a function call equivalent to (cons t nil). Since calling CONS with T and NIL yields a new list whose only element is T, the overall result is (t).
(eval (eval (list 'cons t nil)))
The inner eval again produces (t), and the outer eval then attempts to evaluate this result as a function call by treating T as the function. Because T is not bound as a function, this causes an error (typically reported as “T is not a function”).
(apply #’cons '(t nil))
This expression uses APPLY to call CONS on the argument list (t nil), which is the same as calling (cons t nil). Consequently, the result is the list (t).
(eval nil)
Since NIL evaluates to itself in Lisp, this expression simply returns NIL.
(list 'eval nil)
This expression builds a list containing the symbol EVAL and NIL, so it evaluates to the literal list (eval nil).
(eval (list 'eval nil))
Here a list is constructed as (eval nil) and then immediately evaluated. Evaluating (eval nil) is the same as evaluating NIL, hence the final result is NIL.

2. Arrays and records.
5. The hardware was so slow that the extra overhead of interpretation was negligible, and higher-level programming made debugging and development much easier.
6. The IBM 704 was the first computer to include hardware-based floating-point arithmetic. This innovation allowed numerical operations to be executed directly in hardware rather than via slow software routines, making scientific and engineering computations much more efficient. As a result, high-level programming languages like FORTRAN were developed to leverage this capability, fundamentally shaping the evolution of programming languages by emphasizing built-in, efficient numerical processing.
7. 1954
8. At the time Fortran was designed, computers were primarily used for scientific computations. Early computers were predominantly engaged in handling complex numerical tasks and advanced calculations for research and engineering applications, and Fortran was developed specifically to address these needs.

9. They all came from the plain GOTO—Fortran I's branching constructs were simply different variations of the IBM 704's GOTO instruction.
10. The most significant enhancement from Fortran I to Fortran II was the addition of user-written functions and subroutines, which enabled modular and reusable code and greatly improved procedural programming
11. Fortran 77 introduced block IF statements (i.e., IF with an optional ELSE clause) and logical loop control statements compared to Fortran IV.
14. Linguists were drawn to artificial intelligence because it offered a new way to approach the challenges of natural language processing. They saw potential in using computational methods to model and analyze language structure, an idea that aligned with emerging theories like Chomsky's generative grammar, which aimed to formally describe how language works.
15. Lisp was developed at the Massachusetts Institute of Technology (MIT) by John McCarthy in the late 1950s as he sought to create a language tailored for artificial intelligence research.
20. The missing element was the lack of input/output statements (I/O facilities) in its definition. Without built-in I/O capabilities, vendors had to create their own solutions, leading to incompatibilities and hindering ALGOL 60's widespread adoption.
21. The syntax of ALGOL 60 was described using Backus–Naur Form (BNF), developed by John Backus and later refined by Peter Naur.
22. COBOL was based on Rear Admiral Grace Hopper's FLOW-MATIC programming language. This influence is evident in COBOL's English-like syntax and structure, which were designed to make the language accessible to business users.
23. The COBOL design process began in 1959
24. The data structure was the hierarchical record structure. Plankalkül introduced the idea of grouping related fields within a single record, a concept that later influenced COBOL's design for handling structured business data.
25. The U.S. Department of Defense was most responsible for COBOL's early widespread adoption.
36. A nonprocedural language (also known as a declarative language) is one in which you specify what you want as the outcome rather than detailing how to achieve that outcome. In such languages, you define the desired results, and it's the system's responsibility to determine the steps required to produce them.
37. The two kinds of statements that populate a Prolog database are facts and rules.
46. Interactive television.
51. JavaScript is most widely used for client-side web development—creating interactive and dynamic web pages in browsers
52. JavaScript and PHP are complementary languages in web development. JavaScript runs in the browser, handling client-side interactivity and dynamic behaviors, while PHP processes server-side tasks like database interactions and content generation. They are often used together to create fully featured web applications.
57. In C, cases in a switch statement can fall through to the next one if a break is omitted, often causing unintended behavior. C# addresses this deficiency by disallowing implicit

fall-through; every case must explicitly end (using break, return, throw, or an explicit jump), which makes the control flow clearer and reduces bugs.

59. Xml document and XSLT stylesheet
60. An XSLT processor transforms XML documents using an XSLT stylesheet's rules to produce one or more output documents. This output document can be in various formats, such as XML, HTML, plain text