1. Output with System.out.print() and System.out.println()
   1. An expression that evaluates to a number will be output as a number
   2. An expression that evaluates to a Boolean will be output as a Boolean (true, false)
   3. An expression that contains String concatenation will evaluate to a string
   4. If mathematical types are mixed, the expression will evaluate to the type with the greater precision (int + double = double)
2. Escape sequences are inserted in strings for special character prints, like tabs, backslash, and newline
3. Integer division is in whole numbers only and never returns any fractional part when evaluated
4. Modulus operations (a % b), return only the REMAINDER of a/b
5. JAVA identifiers (class names, method names, and variable names)
   1. Syntax requirements:
      1. must contain only letters, numbers, underscore and dollar sign characters
      2. may not start with a number
   2. Conventions:
      1. the first letter of Class names must be capitalized
      2. the first letter of method names and variables must be a lower case letter
      3. for identifiers created by joining multiple words, every word after the first must have its first letter capitalized
      4. constants (variables declared with the keyword “final” must be in ALL UPPERCASE LETTERS
6. Static Methods in JAVA
   1. any number and types of parameters can be passed from a calling function.
      1. These are local variables in the method, but values are copied from caller
   2. can return any type of value if declared in the Method Header
   3. any method can call any other method of a class
7. JAVA programs process data using three basic methods
   1. SEQUENCE – statements are executed in the order written
   2. REPETITION – loop structures allow a sequence of JAVA statements to be executed a fixed number of times, or as long as some loop condition remains true
   3. SELECTION – selection structures like if/else allow a program to select alternate paths of execution based on the value of some Boolean expression
8. Repetition Structures:
   1. For-loops are intended to implement Definite Loops
      1. uses an initialization section (executed exactly once)
      2. uses a loop-condition such that the loop executes so long as the condition evaluates to “true” (evaluates at the beginning of every loop)
      3. uses an increment section that can increment local variables (executes at the end of every loop cycle)
   2. while-loops are intended to implement Indefinite Loops
      1. uses only a test section that test some Boolean condition at the beginning of every loop and process the loop-code, only if the condition expression evaluates to “true”
      2. The code in the loop body may never execute if the loop condition is initially “false:
      3. Input values that are used to terminate loops are called SENTINEL Values
   3. do-while loops are essentially the same as while-loops except that the loop condition is tested at the end of the code block. This means that the body of the loop ALWAYS executes at least once.
9. Selection structures:
   1. If structures are intended to allow a programmer to determine whether some code block is executed based on some Boolean test condition. If the condition expression evaluates to “true” the code block following is executed
   2. If/else structures provide either-or code blocks. If the Boolean condition of the if statement is true, the if-block is executed, otherwise the else-block is executed
   3. Multiple paths can be defined using if/if else/…/else. In these structures, exactly one code block will execute and the rest will be ignored.
10. JAVA math operators
    1. ( ), [ ], { } (Grouping operators, highest precedence, evaluate contents, left to right)
    2. !, ++, -- (Unary operators for negation, increment, and decrement)
    3. \*, /, % (multiplicative operators, next highest precedence, evaluate left to right)
    4. +, - (additive OR concatenation when a string is one of the arguments, lowest  
        precedence, evaluate left to right
11. Boolean expressions use Relational operators (lower precedence than math, but greater than logical operators)
    1. > >= < <= (less than or equal to)
    2. == != (equality operators, lowest precedence of relational operators)
12. Boolean expressions can be joined with Logical Operators
    1. && - (an expression of the form <Boolean exp1> && <Boolean exp2> is true, only if both exp1 and exp2 are true
    2. || - (an expression of the form <Boolean exp1> || <Boolean exp2> is true, if either exp1 OR exp2 are true (Lower precedence than &&)
    3. ! - (represents the negation of any Boolean expression such that !exp1 it true, if exp1 is false and !exp1 is false if exp1 is true
    4. Short-circuit evaluation – the compiler evaluates logically joined Boolean expressions from left to right. If the left element is false in a compound statement joined with && the whole expression is evaluated to false WITHOUT evaluating the right side. Likewise, if the left side is true in a compound statement joined with || the expression evaluates to true without evaluation of the right hand element
13. Assignment operator = += -= \*= /= %= &&= ||= (lowest precedence of all operators BECAUSE the right side of any expression needs to be fully evaluated BEFORE its value can be stored in the variable on the left side
14. Built-in Math Class methods
    1. Math.min(a, b) returns minimum of a and b
    2. Math.max(a, b) returns maximum of a and b
    3. Math.sqrt(a) returns square root of a
    4. Math.pow(a, b) returns a raised to the b power
15. Built-in String methods
    1. For any defined string, using the dot operator, you can access methods like length(), charAt(), toLowerCase() etc.
    2. For String s, a to get the length use the call s.length()
    3. For a string literal like “Hello World” a call like “Hello World”.length() will return the integer 11
16. Input with the Scanner class
    1. A Scanner variable allows a program to receive input from the keyboard with methods of the Scanner class like:
       1. next() // next string
       2. nextInt() // next integer
       3. nextLine() // next whole line as a string
       4. hasNextInt() // returns true if there is an integer waiting in the keyboard buffer
    2. Scanner variables need to be declared and INSTANTIATED with:  
       Scanner scannerName = new Scanner();  
         
       Then methods calls look like: scannerName.nextInt();