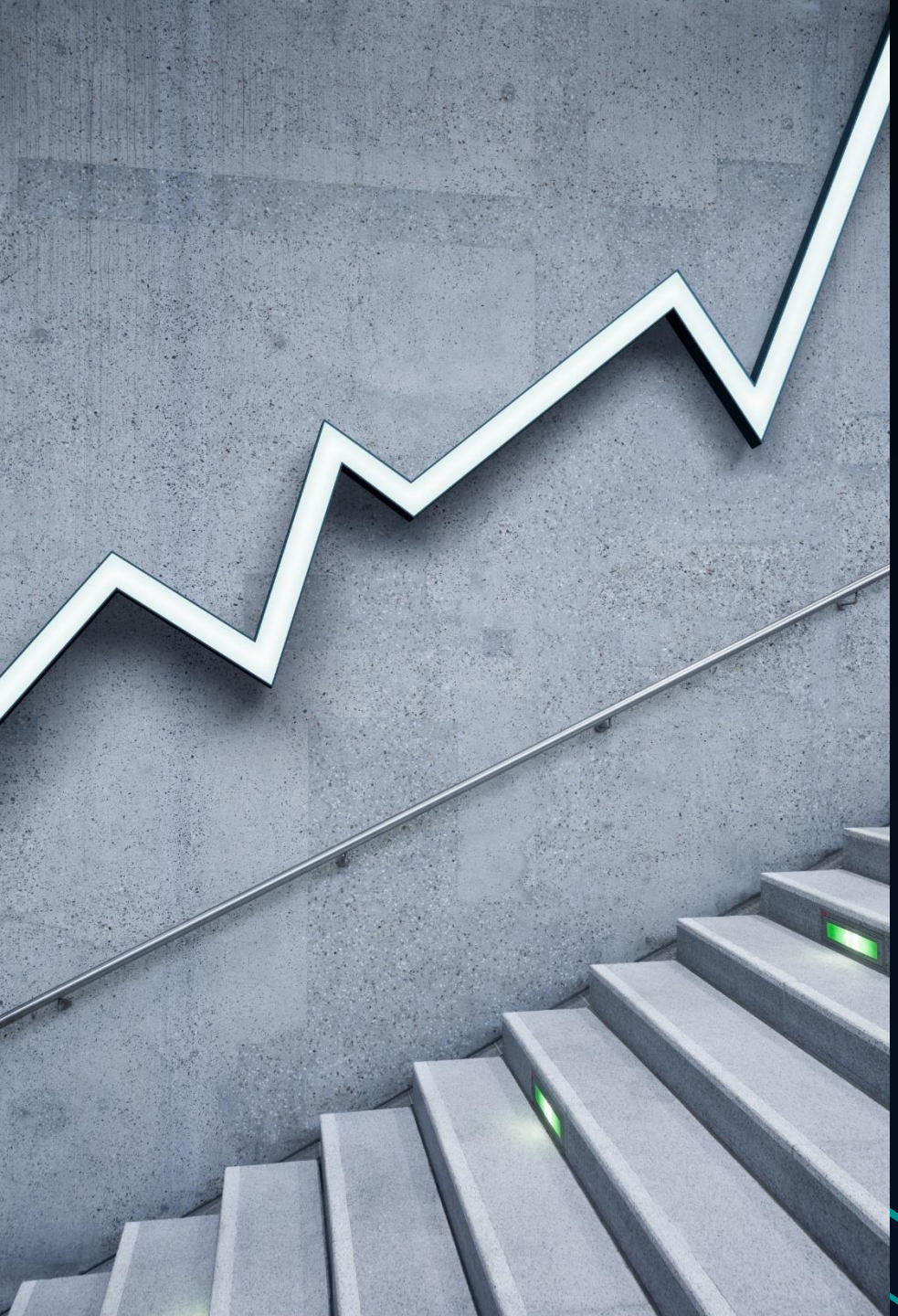


CSI 120 Week 2

LECTURE NOTES

“Programs must be written for people to read, and only incidentally for machines to execute.”

- HAROLD ABELSON

Introduction to Variables

- A **VARIABLE** is a named memory location.
- Variables are **DECLARED** by giving the variable a name and a **DATA TYPE.**
- The **DATA TYPE** determines what type of information can be stored in the variable.

Variable Name

Data Type

```
int counter;
int total;
int x, y, z;
```

Declaration

Multiple variables can be declared in one statement

Variable Names

Variable names may contain letters or numbers, but may not start with a number

- int num1; **Correct**
- int 1num; **Incorrect**

Variable names may not contain a space or any special characters except for the underscore (_)

- string _firstName; **Correct**
- string first Name; **Incorrect**
- string #firstName; **Incorrect**

Should use camelCase meaning that the first letter of the first word is lower case. Additional first letters of words are uppercase

- string lastName;
- int totalClassCount;

Variable Initialization

- A variable must be **initialized** (given a value) before it can be used.
- A variable can be **initialized** at the same time it is declared
  - `string firstName = “Gianni”;`
  - `int num1 = 10;`
  - `int x=5, y=10, z=2;`
- A variable can also be given a value after it has been declared. This is called **assignment.**
  - `int number;`
  - `number = 10;`

The Assignment Operator =

- In C# the = symbol is used for assignment.
  - `int x = 5;`
    - The = in this equation **Assigns** the value 5 to the variable x
  - `double average = 5.9;`
    - The = in this equation **Assigns** the value 5.9 to the variable average

Common Numeric Data Types

- **int –** int (Integer) is used for whole numbers or numbers without a decimal point
  - `100, 5, -1254 are of type` **`int`**
- **double –** double is used to store numbers with decimals
  - `19.99, 2.1, -5.0 are of type` **`double`**

More Common Data Types

- **string –** stores text such as “Hello World”. string values are always surrounded by quotes. This means that the data is words and not numbers. For example, “5” is a string and not an int.
  - “Seahawks”, “Mohamed”, and “22” are of type string
- **char –** stores a single character and is surrounded by a single quote.
  - ‘A’, ‘0’, ‘u’ are of type char
- **bool (Boolean) –** stores a value of either true or false only. Cannot hold any other value. For a statement to be a bool it must be able to evaluate to either true or false
  - 1 > 0 is a Boolean value

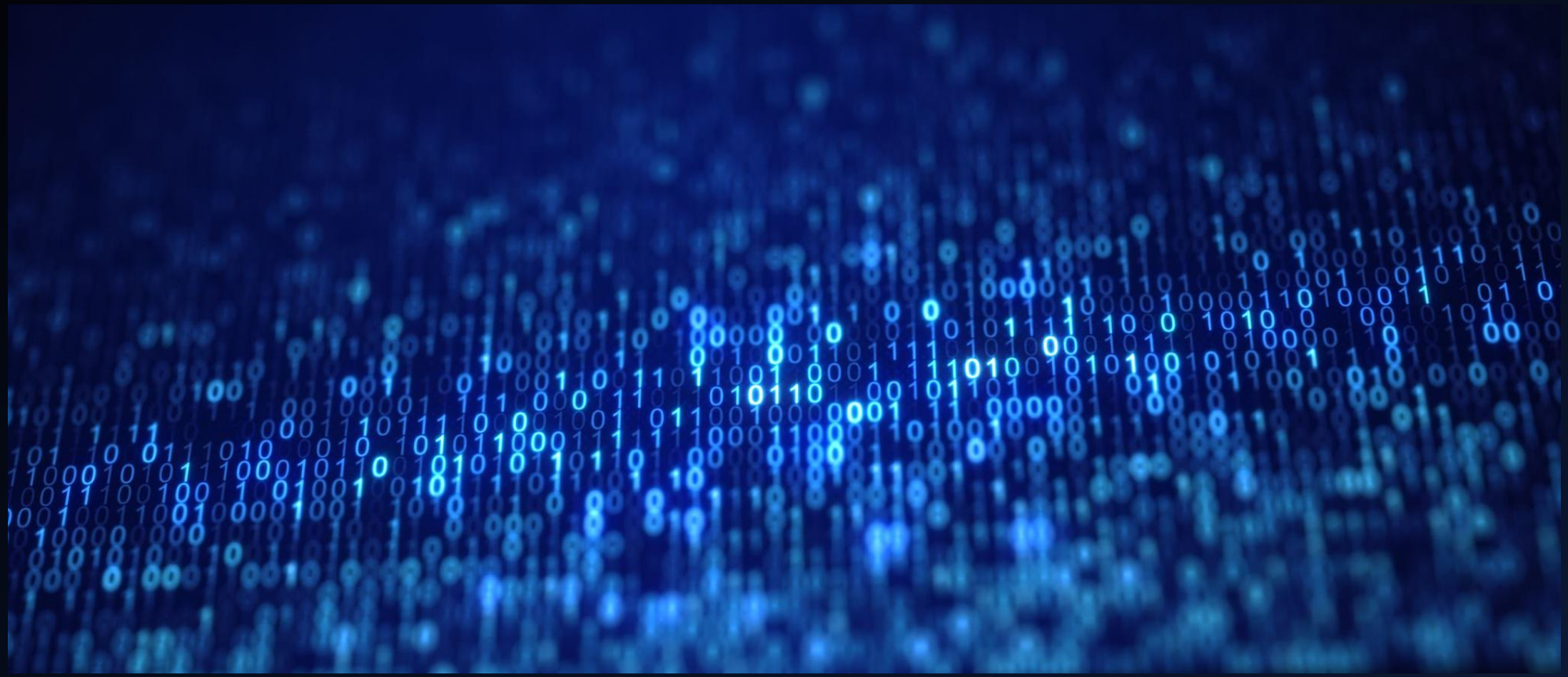
Common Data Type Practice

- What is the common data type of the following values?
  - ‘Z’
  - 5.2123412
  - “Programming I”
  - 10
  - true
  - “110”
  - 2 > 10
  - 10.0
  - “false”
  - -999999999
  - ‘5’ > ‘1’

Output in C# Console Applications

- To send information to the user we use the following syntax:
  - Console.WriteLine(“Information to be sent”);
- The console is how we communicate with the user in C# Console Applications.
- Variables can be output to the console in C# using either Console.WriteLine() or Console.Write(). Note that there is no quotes around the x or firstName in the examples.
  - Console.WriteLine(x); - Prints the value of the X variable to the console
  - Console.WriteLine(firstName); - Prints the value of the firstName variable to the console

LET’S CODE IT! - IntroToVariables



Input in C# Console Applications

To get information from the user we use the following syntax:

- string input = Console.ReadLine();

Note that we first must create a variable to capture the input

Console.ReadLine() needs to be assigned to a string variable to be able to use the input.

Using Strings to get Input

- Information that comes from the user via Console.ReadLine() **always comes in the form of a string!**
- To capture that information a string variable needs to be created. Just typing Console.ReadLine() by itself will not capture the user input.
  - `Console.ReadLine();` - This does not save the input in a variable - Incorrect
  - `string input = Console.ReadLine();` Here the text that is typed into the console will be stored in the string variable input -Correct.

String Concatenation

- When two string variables are added to one another it is called String Concatenation. The strings will be combined into one longer string with no spaces.
  - “Hello” + “World” would become “HelloWorld”
  - if firstName == “Josh” and lastName == “Emery” firstName + LastName will be “JoshuaEmery”
  - “20” + “5” would be “205”

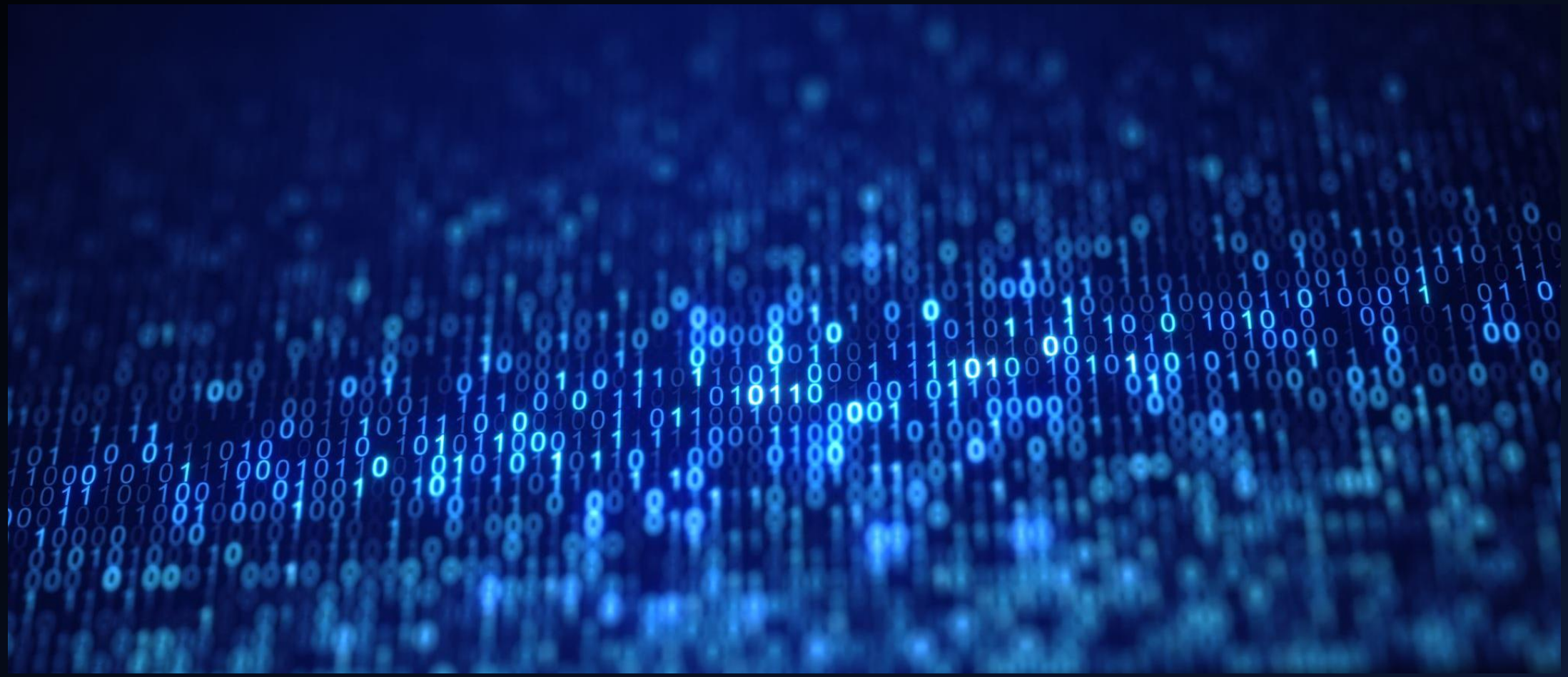
String Concatenation to clarify output

- It is common practice to Concatenate a string with a variable to better explain console output. These examples assume that there is an int number variable set to 10
  - Console.WriteLine(“The number is “ + number); Will output the number is 10.
  - Consle.WriteLine(“The number is “ + number + “ and that is the correct number”); Will output the number is 10 and that is the correct number

Escape Characters

- Escape characters allow you to output special characters that normally would have a meaning to C#, you can also format output with escape characters. Escape characters start with \
  - \n Makes a new Line. Console.WriteLine(“Hello \nWorld”); will output hello and world on separate lines
  - \” Adds a literal “ to the string. This must be done since “ means something to C#. In order to create a string variable that contains a “ escape character must be used.

LET’S CODE IT! - StringInputOutput



Math Operations

- Math can be performed on **Numeric Variables Only**
  - `int sum = 10 * 5; sum would be 50`
  - `double finalPrice = originalPrice – discount;`

Addition	**+**
Subtraction	**-**
Multiplication	*****
Division	**/**
Modulus(Remainder)	**%**

Division Operator /

- If doubles are used for division, then division works exactly as you would expect.
  - `double average = 14.0 / 3.0` – the value of average would be 4.667
  - `double percentage = 8.0 / 10.0` – the value of percentage would be 0.8
- Integer division occurs when one integer is divided by another integer. The resulting fraction will be discarded and only the whole number answer will remain.
  - `double average = 14 / 3` – the value of average would be 4
  - `double percentage = 8 / 10` – the value of percentage would be 0
- If at least one of the operands is a double then the value after the decimal point will not be lost.
  - `double average = 14.0 / 3` – the value of average would be 4.667
  - `double percentage = 8 / 10.0` – the value of percentage would be 0.8

Remainer Operator % modulus

- The Modulus (mod) operator is the remainder operator and gives the **Whole Number** remainder after the first term is divided by the second
  - `int remainder = 14 % 3 – the value of remainder would be 2`
  - `int remainder = 8 % 10.0 – the value of remainder would be 8`
- If the second term divides evenly into the first term, the mod operator will give you 0
  - `int remainder = 4 % 2 – the value of remainder would be 0`
  - `int remainder = 15 % 5 – the value of remainder would be 0`

Convert String to Numeric (Parsing)

- To convert a variable from a string to a number a special type of conversion called parsing should be used. Before user input from Console.ReadLine() can be used for math it must be **Parsed** to a numeric data type.
- Parsing an int from a string
  - `int number = int.Parse("50");` - the value of number will be 50.
  - `int number = int.Parse(input);` - the system will attempt to convert the string variable input to an int.
- Parsing a double from a string
  - `double number = double.Parse("7.42");` - the value of number will be 7.42
  - `double number = double.Parse(input);` - the system will attempt to convert the string variable input to a double.

Data Type Conversion (Casting)

- Casting is converting one data type to another. The data types must be similar for casting to work. You cannot simply cast a string to a number or vice versa.
- Casting a double to an int
  - `int number = (int)5.262` – value of number would be 5
  - `int number = (int)average` – casting can also be used on a variable
- Implicit (automatic) Cast int to a double
  - `double number = 5.` – Here the cast is implicit (automatic) the value of number will be 5 as a double.

Modifying an existing variable

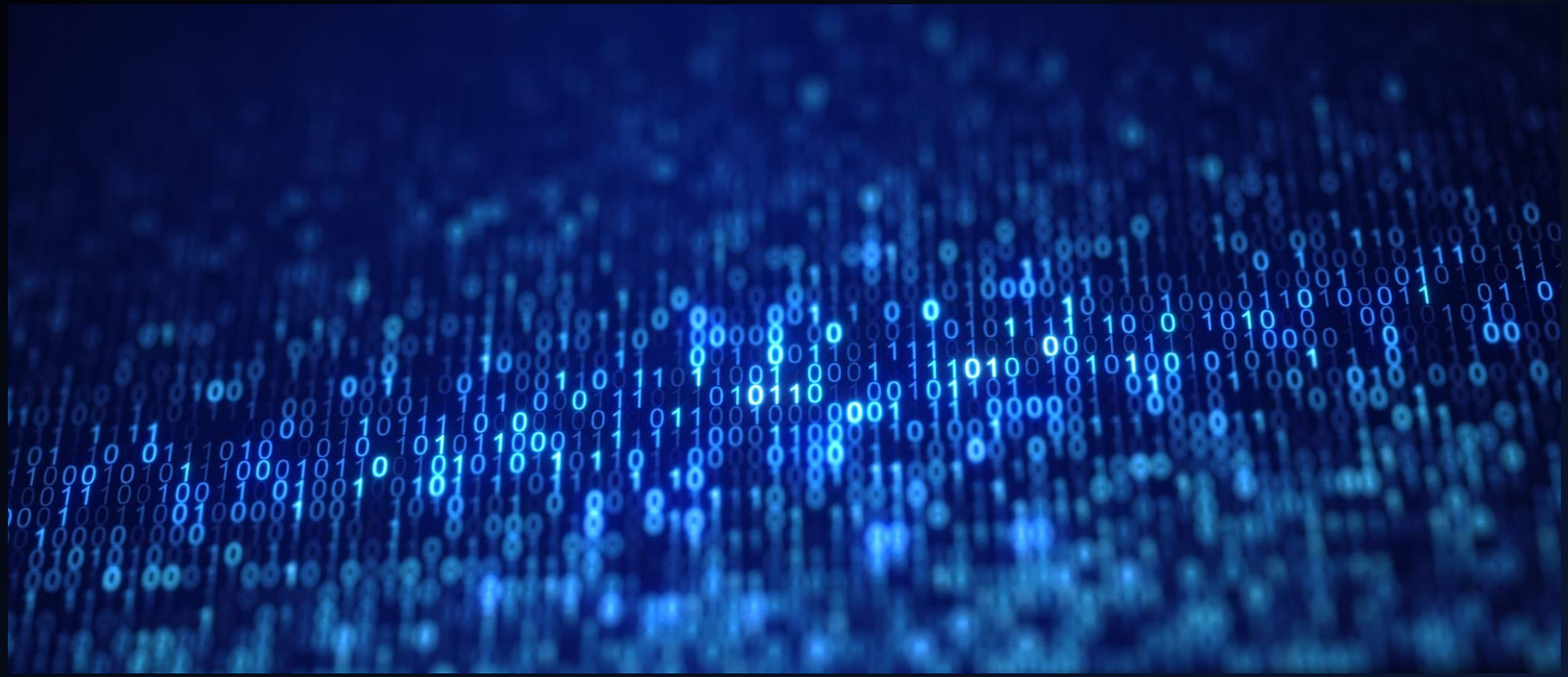
- A variable which already has a value assigned can be used in both sides of an assignment statement. Assume that total = 10 for the following examples
  - `total = total + 1;` - In this example the right side of the equation is evaluated first and 11 will be store back in total
  - `total = total – 1;` - Here the right side of the equation would equal 9, so 9 would be stored in the total variable.
  - `total = total * 3;` - In this example the right side of the equation would evaluate to 30 and 30 would be stored in total
  - `total = total / 2;` - Here the right side of the equation would equal 5 so 5 is stored in the total variable.

Operator Precedence

- Math Operations can be combined into complex expressions
  - `double result = total + count / max – offset;`
- Operands have a well-defined priority that they follow

1. Parentheses ()
2. Multiplication *, Division /, Remainder %
3. Addition +, Subtraction -, String Concatenation +
4. Assignment

LET’S CODE IT! - MathandDataConversion



Operator Precedence Practice

```
int a=3, b=5, c=2;

int answer;

answer = a * b - c;

answer = b + a * c;

answer = b / a;

answer = a / b;

answer = b % a;

answer = a % b;

answer = b - a * b - c;

answer = b - a / c;

answer = (b - a) / c;

answer = a * (b + c);
```

Given the following declarations, what result is stored in each of the statements?

```
int iResult, num1=25, num2=40, num3=17, num4=5;
double dResult, val1=17.0, val2=12.78;
```

1. `iResult = num1 / num4;`
2. `dResult = num1 / num4;`
3. `iResult = num3 / num4;`
4. `dResult = num3 / num4;`
5. `dResult = val1 / num4;`
6. `dResult = (double)num1 / num2;`
7. `dResult = num1 / (double)num2;`
8. `iResult = (int)(val1 / num4);`
9. `dResult = (int)((double)num1/num2);`
10. `iResult = num3%num4;`

Arithmetic Operator Practice

1. `10 + 3`
2. `3 * 5`
3. `5 - 2 * 3`
4. `3 / 2`
5. `5 % 3`
6. `2 * 3 + 4 / 5`
7. `3 * 4 / (6 - 5 % 2)`
8. `1 + 2 * 3 / 4`
9. `5 % 6 - 2`
10. `3 + 2 * (2 – 5) / 4`