Overview (1):
- **Before We Begin**
  - Some administrative details
  - Some questions to consider
- **Data: Variables and Constants**
  - Data types & Visual Basic (cont. from last lecture)
  - Declaring variables and constants
  - Variable scope
- **Data Types**
  - Working with data types

Overview (2):
- **Arithmetic Operators**
  - Allowable arithmetic operators
Before We Begin

Administrative Details (1):
- Lab Exercise 3-3
  - You should be working on Exercise 3-3 this week!
  - I will drop by the Glade lab Friday after the lecture for about 30 minutes

Some Questions to Consider (1):
- What is a variable?
- What is a constant?
- Why do we need variables?
- What is a variable's type?
- What is a variable declaration?
Data: Variables and Constants
(cont. from last lecture)

Data Types and Visual Basic (5):

- Visual Basic Variable Data Types (cont.)
  - Most common type of variables and constants (at least in this course)
    - String, Integer, Boolean, Double
  - Of course, it is up to you as a programmer to determine the variable type but some common guidelines are as follows
    - If data is used in a calculation → numeric type
    - If not used in a calculation → String
    - Scientific calculations → Single or Double

Data Types and Visual Basic (6):

- Visual Basic Variable Data Types (cont.)
  - Consider the following examples

<table>
<thead>
<tr>
<th>CONTENTS</th>
<th>DATA TYPE</th>
<th>REASON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Social security</td>
<td>String</td>
<td>Not used in calculation</td>
</tr>
<tr>
<td>Pay rate</td>
<td>Currency</td>
<td>Used in calculation - represents money</td>
</tr>
<tr>
<td>Hours worked</td>
<td>Single</td>
<td>Used in calculations and may contain decimal</td>
</tr>
<tr>
<td>Phone number</td>
<td>String</td>
<td>Not used in calculations</td>
</tr>
<tr>
<td>Quantity</td>
<td>Integer</td>
<td>Used in calculations but generally whole number</td>
</tr>
</tbody>
</table>
Data Types and Visual Basic (7):
- Naming Conventions and Rules
  - It's up to you as a programmer to provide the names for the variables and constants you declare
  - VB requirements
    - 1-255 characters long
    - Letters, digits and underscore characters only → no spaces or periods
    - May not be reserved words!
  - Aside from VB requirements, the main thing is to be consistent!

Data Types and Visual Basic (8):
- Naming Conventions and Rules (cont.)
  - Provided you follow the VB rules, you are free to choose any name you want → still have some general conventions we try to follow so that we can separate good names from bad names
    - Choose meaningful & descriptive names → a name should indicate the variables purpose
    - Precede each identifier with a lower case prefix
    - Capitalize each word of the name following the prefix → always use mixed case, never all upper case (e.g., myIntegerValue)

Data Types and Visual Basic (9):
- Naming Conventions and Rules (cont.)
  - Some "good" (descriptive) variable name examples

<table>
<thead>
<tr>
<th>FIELD OF DATA</th>
<th>POSSIBLE IDENTIFIER (NAME)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Social security number</td>
<td>socialSecurityNumber</td>
</tr>
<tr>
<td>Pay rate</td>
<td>payRate</td>
</tr>
<tr>
<td>Hours worked</td>
<td>hourWorked</td>
</tr>
<tr>
<td>Phone number</td>
<td>phoneNumber</td>
</tr>
<tr>
<td>Quantity</td>
<td>quantity</td>
</tr>
<tr>
<td>Tax rate</td>
<td>taxRate</td>
</tr>
</tbody>
</table>
**Data Types and Visual Basic (10):**

- **Naming Conventions and Rules (cont.)**
  - Some "BAD" (descriptive) variable name examples

<table>
<thead>
<tr>
<th>FIELD OF DATA</th>
<th>POSSIBLE IDENTIFIER (NAME)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Social security number</td>
<td>ssn</td>
</tr>
<tr>
<td>Pay rate</td>
<td>a</td>
</tr>
<tr>
<td>Hours worked</td>
<td>w</td>
</tr>
<tr>
<td>Phone number</td>
<td>num</td>
</tr>
<tr>
<td>Quantity</td>
<td>q</td>
</tr>
<tr>
<td>Tax rate</td>
<td>r</td>
</tr>
</tbody>
</table>

**Declaring Variables & Constants (1):**

- **Declaring Constants**
  - Const Identifier [As Datatype] = Value
  - Const → reserved word indicating a constant variable
  - Identifier → user-defined name of the variable
  - As Datatype → indicates the data type and if not included (it is optional) then data type is of type variant
  - Value → the assigned value (should be of compatible type!) and must be provided!

**Declaring Variables & Constants (2):**

- **Declaring Constants (cont.)**
  - Example constant declarations
    - Const courseName As String = "CSE 1530"
    - Const companyAddress = "101 - Main Street"
    - Const salesTaxRate As Single = 0.8
  - Are the following valid? How can we test this?
    - Const myName As String
    - Const todaysDate As Date
Declaring Variables & Constants (3):
- Declaring Variables
  
  Dim Identifier [As Datatype]
  
  - Dim → Dimension (size)
  - Identifier → user-defined name of the variable
  - As Datatype → indicates the data type and if not included (it is optional) data type is of type variant

Declaring Variables & Constants (4):
- Declaring Variables
  
  Example variable declarations
  - Dim customerName As String
  - Dim totalSold As Integer
  - Dim temperature As Single
  - Dim productPrice As Currency
  - Dim changing

  - What is the type of the variable "changing"?
  - Is this declaration valid?

Variable Scope (1):
- The “Visibility” of a Variable
  
  Scope of a variable describes the “visibility” of a variable you declare
  - Visibility → Where the variable exists, can be seen and is accessible to you
  - Can be for the entire project, for only one form or for only one procedure
  - Scope is said to be global, module level or local
Variable Scope (2):
- The “Visibility” of a Variable (cont.)
  - Global variable
    - May be used in all procedures of a project
  - Module level variable
    - Accessible from all procedures of a form
  - Local variable
    - Can be used only within the procedure in which it is declared

Data Types

Working With Data Types (1):
- In General
  - Dangerous to assign a value (data) of one type to an object property or property that should contain data of another type
  - Will typically result in an error → at times, you may not even be aware of the error (may not necessarily result in a run time error but rather, the result computed may be incorrect)
  - Visual Basic will attempt to convert to the proper data type when incorrect data types are assigned but it may not always be converted correctly!
Introduction to Computer Use II

**Working With Data Types (2):**

- **In General (cont.)**

  - Visual Basic will convert conversions between data types whenever it can in a sensible manner.
  - `Text1.Top = "Visual Basic"` → will result in an error.
  - `Text1.Top` is an Integer and "Visual Basic" is a non-alpha-numeric String.

**Error message due to above statement.**

**Working With Data Types (3):**

- **In General (cont.)**

  - `Text1.Top = "335.67"` → no error! Why?
    - `Text1.Top` is still an Integer of course however, the String "335.67" is a string that contains only alpha-numeric characters → Visual Basic can then convert this string (automatically) to an Integer value.
    - But 335.67 is not an integer ??? → Visual Basic will convert the value to an Integer by eliminating (dropping) the decimal portion.
    - The String "1,001" will also be converted to 1001.

**Working With Data Types (4):**

- **In General (cont.)**

  - Not only Strings will be converted → Visual Basic will attempt to convert any "mis-matched" data type when it can in a sensible manner.
  - In fact, typically any other data type can be converted to a String without ambiguity.
  - Examples that will be converted to a String:
    - `Text1.Text = 232`
    - `Text2.Text = True`
    - `Text3.Text = 26.00211`
Working With Data Types (5):
- “Take-Home Message”
  - It is dangerous to rely on visual Basic to convert between data types
  - It may often succeed but there are many times it will not!
  - You should always be aware of the data types you are using and ensure that values have the appropriate types
  - But there are times where we need to convert data from one type to another → User input is typically of type String and must be converted to some value (Integer etc.)

Working With Data Types (6):
- Built in VB Conversion Functions
  - Visual Basic functions to convert between data types
  - As an aside → what is a function?
    - A convenient way to encapsulate some computation that can then be used many times over without worrying about its implementation
    - Allows us to ignore how a job is done
    - All we need to know is what is done (outcome)
    - Imagine having to compute some computation many times → you can replicate the code many times or you can write the code once within a function and simply call the function

Working With Data Types (7):
- Built in VB Conversion Functions (cont.)
  - In general these conversion functions take one or more arguments and produce a single result (called the function return type)
    - Argument → when you call and use the function, you may have to supply it zero or more values - these values are known as arguments
    - Function return type → the value returned by the function - the value can be used by the caller of the function where appropriate
  - More details regarding functions later on in the course
Arithmetic Operators

Arithmetic Operators (1):
- Allowable Arithmetic Operators
  - Addition, subtraction, multiplication, division and exponentiation

<table>
<thead>
<tr>
<th>OPERATOR</th>
<th>OPERATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
</tr>
<tr>
<td>/</td>
<td>Floating Point Division</td>
</tr>
<tr>
<td>\</td>
<td>Integer Division</td>
</tr>
<tr>
<td>^</td>
<td>Exponentiation</td>
</tr>
<tr>
<td>Mod</td>
<td>Modulus</td>
</tr>
</tbody>
</table>

Arithmetic Operators (2):
- Usage of Arithmetic Operators
  - Addition → result = expr1 + expr2
  - Subtraction → result = expr1 - expr2
  - Multiplication → result = expr1 * expr2
  - Division → result = expr1 / expr2 (decimal result)
    → result = expr1 \ expr2 (integer result)
  - Exponentiation → result = expr1^exponent
  - Modulus → result = expr1 Mod expr2 (remainder of after the division operation where expr1 and expr2 are both integers)
Introduction to Computer Use II

**Arithmetic Operators (3):**

- **Usage of Arithmetic Operators**
  - `Command1.Top / 2`
    - Divide the Top property of the Command 1 object in two → keep in mind that the actual value of the Top property of the Command1 object does not change ≠ we are not assigning the result of this arithmetic operation back to the Top property
  - `Command2.Top = Command1.Top / 2`
    - Divide the Top property of the Command 2 object is assigned the value of the Top property of the Command1 object divided by two (e.g., if Command1.Top is 100, then Command2.Top is 50)

**Arithmetic Operators (4):**

- **Order of Operations**
  - The order in which arithmetic operations are performed will affect the final result
  - `3 + 4 * 2` → if addition is performed first then result is 14 but if multiplication is performed first then result is 11
  - Order of precedence in VB arithmetic operations
    1. Exponentiation
    2. Multiplication and division (in order from left to right)
    3. Addition and subtraction (in order from left to right)