

# **INTRODUCTION TO VARIABLES**

## **VARIABLES**

Almost everything you do in programming will require you to use variables. Therefore, the correct usage of variables is necessary to successfully program in any language. When you look through various resources on computer programming, you find discussions of many different types of variables. Data, reference, instance, dynamic, static, constant, local, and global seem to be the ones most commonly discussed. Unfortunately, these types of variables don't have universal meanings. Even when you limit your discussion of variables to the confines of one language, you will still see variations. This can be confusing for a new programmer. My advice is to pick one or two good resources for your language of choice and spend the time learning, fully understanding, and using those terms and concepts. Once you have done that, it becomes much easier to figure out where the similarities and differences lie between resources and languages.

In spite of the variations, there are a few things about variables that most programming resources agree upon

- Variables are related to locations in the computer's memory.
- A variable can only hold one value at any given time.
- The value held in a variable can vary (unless you specifically say it can't).
- Each variable has a unique name.

## **NAMING LIMITATIONS AND CONVENTIONS**

“Some programmers have fun with their variable names by naming them after friends or creating puns with them, but such behavior is unprofessional and marks those programmers as amateurs.”

- Joyce Farrell, *An Object-Oriented Approach to Programming Logic and Design*, Thomson Course Technology, 2008.

Variables have names. In most languages, we can use actual words that have meaning to us to name our variables. Once we properly name a variable, we only have to call the variable by name when we want to use it.

Each programming language has its own rules about naming variables. Certain characters or words may not be used. Name length may be limited. If you break one of these rules, you will get an error and your code will not run. You must name your variables within these limitations.

In addition to the limitations of the language, programmers also follow certain conventions when naming their variables. Some conventions are considered standard practices within a professional community and therefore have shared meaning. Some conventions are dictated to you by your boss, your development team or your clients. Programmers will also develop naming conventions as part of their own personal coding style. Unlike the naming limitations of a language, you will not get an error if you don't follow one of the conventions. However, following conventions may actually be more important. Programmers have crashed a client's server when they did not follow the client's naming conventions. Programmers have stalled development when their code did not work with code written by other team members. Programmers did not get hired because when asked to write code during the interview process they did not follow industry naming conventions.

You may not like them, but you need to follow both naming limitations and conventions.

## **VARIABLE NAMING LIMITATIONS AND CONVENTIONS IN ACTIONSCRIPT 3.0**

### Limitations

- Variable names can not be a reserved word or keyword
- Variable names must start with a letter, underscore, or dollar sign
- Variable names can not use special characters (& or \* or # or a space are examples of special characters)
- Variable names must be unique

## Conventions

- Camel case - variable names start with a lowercase letter and include uppercase letters, for example `myNewVariable`
- Underscores are only used to start names of private variables.
- Dollar signs are only used to start names of static variables
- Variable names should be descriptive so you (or someone else) know what the variable is every time you see it in code

## **DATA VARIABLES**

Data variables are named containers used to store information for later use. Think about data variables as coffee cups and data as coffee. Coffee is much easier to drink and to move around when its in a cup. Although we could drink out of some coffee makers, we would probably make a mess and get burned.

In object oriented programming, data variables are most frequently used in one of three ways: as function (local) variables, as class (static) variables, or as instance variables. Class and instance variables have access modifiers. These three specific uses of variables will be covered later.

## **DATA TYPES**

Computers handle different types of data differently. Therefore, we need to tell the computer what kind of data to expect. That's exactly what data types do – tell the computer what kind of data to expect. In some languages, the data type also specifies the amount of storage needed by the variable. Data types also tell (or remind) the programmer something about the purpose of a variable.

In strongly typed languages, every variable must have a data type when you declare it. Variables can only contain the type of data you've told the computer it can contain. In the beginning, using data types may seem like an extra step that slows you down because you have to plan exactly what your application is going to do and how it will handle data before you start writing code. However, data types are actually very helpful because they help your code be more predictable, your applications be more stable and they are invaluable when debugging your code. Data types are very useful and powerful tools when developing an application.

Still a little unsure about data types? Think about the toy at the right. The toy helps toddlers learn to recognize shapes. Star shaped blocks only fit in the star holes. Cubes only fit in the square holes. Cylinders only fit in the circle holes. The box gives toddlers an immediate error – the block won't fit – when they try to put the star into the square hole. In a similar way, the computer will give you an error when you try to fit the wrong type of data into a variable that's been typed with a different data type.



Each programming language deals with data differently and has its own data types. In general all languages will have at least one data type for numeric data and one for text data. Typically each language will allow you to use many different data types. Here are the ActionScript 3.0 data types we will use most frequently this semester:

- **Boolean** – true or false; default value is false
- **String** – sequence of characters; used for text data; always put in quotes; default value is null
- **Number** – integers, unsigned integers, floating point numbers; default value is NaN
- **int** – 32-bit integer values from -2,147,483,648 to 2,147,483,647; default value is 0
- **uint** – 32-bit unsigned integer with values from 0 to 4,294,967,295; default value is 0

## ACCESS MODIFIERS

Access to variables, classes, functions, and methods are controlled using access modifiers. Encapsulation is a very important concept in object oriented programming that you will learn about throughout the semester. Access modifiers help achieve encapsulation.

These keywords are used when declaring variables to control access in ActionScript 3.0

- **public** – variable is available to all classes
- **private** – variable can only be accessed by the class in which the variable is defined
- **protected** – variable can only be accessed by the class in which the variable is defined and by any subclass
- **internal** – variable can only be accessed by classes within the same package as the class in which the variable is defined

Don't worry about classes yet. You will learn that soon. For now, just use **public** when declaring your variables

## VARIABLE DECLARATION

Creating a new variable is called variable **declaration**. In most programming languages, variable declaration must include at least the name of the variable and the data type the variable will contain. Some languages also require you to use a keyword that tells the computer you are declaring a variable. ActionScript is one of those languages – you must use the keyword **var**. There are four parts to (non-local) variable declaration in ActionScript 3.0

1. access modifier
2. keyword telling computer to create variable
3. variable name
4. type of data to be contained in variable

Generic variable declaration:

```
accessModifier var myVariableName:DataType;
```

Examples of variable declaration

```
public var userNameInput:String;  
private var correctAnswer:Boolean;  
public var userAge:uint;  
public var predictedTemp:int;  
private var productPrice:Number;
```

When declaring a local variable, you do not use an access modifier. Local variables will be discussed with functions and methods.

## DATA ASSIGNMENT

We use variables to hold data. Putting data into a variable is called **assignment** and uses the assignment operator **=**. We use the assignment operator with a variable name on the left and the data to be put into the variable on the right. You can assign data anytime after a variable has been declared.

Generic variable assignment

```
variableName = data;
```

The following example declares a variable named **userInputAge** to hold an un-signed integer and then assigns it the value of 21. There can be many lines of code separating declaration and assignment.

```
public var userInputAge:uint;  
userInputAge = 21;
```

Each of the variables below were previously declared. Here we are assigning data into each of them

```
userNameInput = "Jane";
correctAnswer = true;
userAge = 19;
predictedTemp = -5;
productPrice = 5.99;
```

Because you can assign data anytime after a variable has been declared, you can do it immediately after declaration – literally in the same statement.

```
public var userNameInput:String = "Jane";
private var correctAnswer:Boolean = true;
public var userAge:uint = 19;
public var predictedTemp:int = -5;
private var productPrice:Number = 5.99;
```

Assignment can take place at almost any time after a variable has been declared – at declaration, many lines later, or after an event occurs (i.e. the user typing in data or the system loading external data). When assignment takes place is usually determined by when you know the data.

## **SCOPE**

Scope refers to the area of the code where the variable is recognized. A variable's scope is determined by where in the code the variable declared. Variables declared within a block of code are only available within that block of code.

## **CONSTANTS**

Constants are data variables whose value can't change after it has been assigned. You use constants to hold data that will never change or data that needs to be protected from change. If you (or the program) attempts to change the data in a constant, you will get an error message.

Constants are declared using the `const` keyword rather than `var`. Constant names are usually all caps with words separated by an underscore

Example: if you are building an application that will convert miles to kilometers, you would use a constant variable to hold the value that you must multiply miles by to get kilometers. If that value were to somehow change, your conversions would no longer be correct. The following statement declares a constant variable named `MILES_KM_MULTIPLIER` to hold Number data and assigns it the value of 1.61

```
const MILES_KM_MULTIPLIER:Number = 1.61;
```