**Variables**

All programming code consists of command statements that work on data. Data can be a highly complex structure which is often held in a database or a simple number or word. To become a good programmer you must understand how data works. The basic building block of data is the variable.

Variables are letters or words that stands for the data in the code of your program. Remember when you went to high school maths and learned Algebra. I’m sure that when you saw:

x = 3

y = 4

z = x + y

you thought (like I did) why do I have to use the letter x when I can just use a three???? Why do I need to learn this stuff??? Where will I ever need it??? Guess what, now you need it.

The actual data (like 3 in the example above) is called a **literal** in programming terms. Using literals in programs is a very bad idea. The only place you should use a literal is when you are declaring a variable that you know about. They should never be used in the body of your program.

Imagine that you are a professional programmer being paid $1000 an hour for your time (I know that ridiculous, no-one gets paid that much to be a programmer do they Bill Gates) and you have created a banking program that is hundreds of thousands of lines of code. In the code you often need to add GST which is a flat 10% so you have put the value 10 into hundreds of calculations. The Government has just changed GST from 10% to 11.5%, so the banks come to you and say they want the program changed to the new value and since it’s a small change you only charge for one hour’s work. Being a good programmer you used a variable for the GST, so to make the change you simply change one line of code, 2 minutes’ work, $1000 please.

However, if you had use the literal 10 rather than the variable GST, your job would now be to go through every line of code working out whether the 10 was GST or something else and changing only those that were correct. Now you would need to do hours of testing to ensure that every transaction actually used the new value, 2 month’s work, $1000 please.

**So what do variables look like?**

Variables all have a name, the name can be a single letter or words. However you cannot use spaces in your variable names so if you want to name a variable my variable you will need to push all the words together myvariable. This does get confusing so many programmers use the underscore to indicate the white spaces and using all lower case letters so myvariable becomes my\_variable. This is the preferred method for Python programming. Usually by convention the first letter of a variable name is in small letters. Some languages (like PHP) add a character to variables, so in PHP my\_variable would be $my\_variable. The major alternative to the underscore is camel case when doing this (in camel case every word is capitalised) so myvariable is written myVariable.

Programmers go to great lengths to find good names for their variables, which makes their programs easier to read by other programmers (the computers don’t actually care), however it is important not to go overboard with name (this\_value\_is\_used\_to\_calculate\_the\_gst is a good variable name but the same could be done by using the name gst\_value or GST\_value). There are some other conventions around names in programming, but we will cover those when we get to them.

**What are the various types of variables?**

Not all data is the same and computer programs need to know what type of data they are getting and usually get upset when they expect one type of data and you use another (when programs get upset they give you an error message). The basic data types are:

|  |  |
| --- | --- |
| Numbers | Like 3, -45, 6.5 etc However a computer programs often have various sub categories of numbers:**Integers** are whole numbers like 6 or -34, some languages go even further and have a different data type for small integers and large integers (eg Java, C).**Decimals** are numbers which include a decimal point like 3.5. These are often call **Floats.** Be careful not to mix integers and floats in the one calculation (3 + 4.5), in this situation you will need to change them to floats. Some languages (like Python) make this change automatically for you**Money** is a special category of decimals. You can’t use floats for money so in most cases computer languages build a whole special data structure for money. The problem with the float data type is that it is not accurate.**Time and Date** is also a special category of number and again is usually covered by special data structures**Binary** is often used in programming and you may find special data types for binary and hexadecimal numbers.Python has four data types for numbers: Integer, Float, Long and Complex. In this course we will be using just Integer and Float. |
| Text | Used when you want words to appear on the screen. Computer languages have a special term for text:**Characters** are individual letters (like a). Sometimes they have a data type all of their own.**Strings** are sequences of character used to form words and sentences.Python has one data type for text: String. |
| Boolean | This is a data type which can only hold two values, true or false (which is sometime represented by 1 and 0). This data type is very useful for developing the flow of a program.Python has one data type for Boolean: There are three keywords True, False and None |
| Array | This type is used to group other data types together. It is such an important data type we will be spending some time looking at this type.Python has many data types for Array: List, Set, FrozenSet, Dictionary, ByteArray and Tuple. In this course we will cover just the List type. |
| Other Data types, not present in all languages | **Constants** these are often not a separate data type but usually a special declaration of an existing data type to show that the value cannot change. GST in the example above would be a good candidate to be a constant. Often you name constants using all capitals to show in the name that they are a constant.**Class** in Object Oriented programming languages you can create your own data type which can be as complex as you like.**Language specific data types**, when you learn a language the first thing you need to do is learn what data types are available to that language, many languages have data types specific to the language eg Python has complex (for complex numbers), C++ has wchar\_t (for characters 2 or 4 bytes wide), PHP has Null (which is used in memory management), Java has a primitive (its name for data types) called byte (which covers the range -128 to 127). |

**How do you assign the data to a variable?**

All programming languages have a method for assigning the data (either a literal or developed at run time) to a variable before the variable is used elsewhere. This process is often called **declaring the variable**. However there are two basic types of methods which are often used to categorise languages.

**Static Type languages**

These languages (like Java, Go, Pascal, and C) all require you to specifically declare the type of variable before you use it. This allows the language to set aside the correct amount of memory for variable (obviously a String could use much more memory space than an integer). These languages also throw an error if you later try to use that variable name with a different type of variable.

Static type languages are often thought of as ***type safe*** because it catches all the type errors early in the development process. In these languages the process of declaring the variable also includes a declaration of the type of variable which will be assigned to that name. Once declared that variable name will only accept variables values of that type. eg.

public int x;

in Java says that the variable called x is of the type int (integer) and can be used anywhere in the program (public, see the discussion on scope below). After this

x=x+1 will work

x = x +.5 will throw an error because .5 is a literal float and x can only hold integers

(In Java there is a way to add an integer and float together to produce a float as an answer, however it is quite an involved process).

**Dynamic Type languages**

These languages (like Python, Ruby, PHP and Javascript) all allow you to use the variable names however you want. This allows you to reuse variable names to suit what you are doing at the moment and there are no checks during development to see whether the variable will actually work properly.

One advantage of dynamically typed languages is they are far more flexible in their use of data developed during run time, which makes programming databases using these types of languages often far easier. Eg. In PHP

$x = 1;

$x = $x +1;

$x = $x + .5;

works fine, in the first line $x has the value of 1 and is of the type integer, after the second line the value of $x is 1.5 and it now has the type float.

**Variable Scope**

This concept refers to where in the program you can use a variable. Obviously the variable can be used in the place where it is declared, however can it also be used in other parts of the program? The answer to this depends on its scope.

In some statically typed language, part of the declaration process includes declaring the scope of the variable (see above the keyword public in Java). In dynamically typed languages the place where the variable is declared also determines its scope.

As a rule of thumb it is often best to declare a variable close to where you want to use it, however this may give the variable the wrong scope. In those cases you declare the variable where it can have the correct scope and then often re-assign the variable close to where it will be used.

**Using Variables**

Variables are used throughout the program in exactly the same way that the literal would be. However when they are used as part of functions there may be some issues in how the language treats the original variable and the passed data. (In this example I will assume I am working with a dynamically typed language, since a static type language would require that x,y and z be declared first before they can be assigned values) Eg

z=10;

x = z;

y = multiply\_by\_hundred(x);

after this, what is the value of z, it will vary from language to language depending on how the language deals with the issue of passing the value of x into the command multiply\_by\_hundred(). So let’s go through how the computer language actually handles each line.

z=10;

z is a variable. The literal 10 is an integer value so the variable z has a value of 10. This means that the computer language will allocate an area of memory to the variable z and place the value of 10 into that area of memory. It will also develop a **table of variables** and the variable z will have a pointer attached to it pointing to the area of memory which has been allocated to it holding the value of 10 currently.

x=z

x is a variable of the type integer, however what is the value that is placed in the area of memory set aside for x? Since x has not received a literal, the area of memory actually has no value at present. The table of variables however now contains both x and z and both are pointing to the same area of memory to get the value, the area of memory that has been assigned to z. What this means is that the **reference** to 10 was passed to the variable z, not the actual value of 10. The process is called **passed by reference**. For dynamic type language this can be very memory efficient since at this stage it does not even have to set aside an area of memory to hold the value of x.

y = multiply\_by\_hundred(x);

y is a variable of a type determined by the output of the command multiply\_by\_hundred(). The value of y is the outcome of the process of the command. However what happens to x and z during the process, are they changed as well? The answer is it depends.

**Pass by Reference, Pass by Value**

When you see multiply\_by\_hundred(x), what is happening with the x. It is a variable of the type int which is currently referencing the value of the variable z. Computer languages have two choices on how to deal with this, they can either pass the reference to the command or they can pass the value to the command. How they handle this situation is determined from language to language and part of learning new language is learning how each one works. Eg.

Java always passes the value, which means that the x in multiplyByHundred is immediately replaced by the literal so it becomes multipleByHundred(10). This means whatever happens in mutiplybyHundred, cannot touch the original variable of x (or z).

PHP usually passes by value, however it can pass by reference by using the symbol & thus mulitplyByHundred($x) would pass by value while multiplyByhundred(&$x) would pass by reference. If you pass by reference then multipByHundred could change the value of x(and z) not just within the command but also back to the original variable.

VBScript usually passes by reference, however it can pass by values by using the keyword ByVal when the command is created.

**Mutability**

The ability to change variable values inside a function without reassignment (without using the = sign) them has a term, it is called **mutability.** In Python integers, floats, Strings etc are all immutable, they cannot be changed unless you reassign them; while lists, tuples and dictionaries etc are all mutable, they can be changed without reassigning them.

**Confused?**

If not then stop reading, you are already a killer programmer and you need to complete your Doctorate of Computer Science. These concepts are hard and will take time. The best way is to experiment with your current language (or your latest language) to find out exactly how it handles variables in a number of circumstances. Whenever I am learning a new language I always develop a few small programs where I can test out commands, variable etc before I try to use them in my programming projects.

Glossary

|  |  |  |
| --- | --- | --- |
| **Variables** | Literal | The actual data that is being used |
|  | Integer | A number type that refers to whole numbers including negative numbers |
|  | Character | A single letter |
|  | String | A sequence of characters |
|  | Boolean | True or False |
|  | Array | Special data type structure |
|  | Declaring | The method of setting up a variable with its name, scope, variable data type and its value. |
|  | Static-Type Language | Computer languages that insist that all variable names have a data type attached to them which does not change |
|  | Dynamic-Type Languages | Computer languages which allow you to change the data type of the variable dependent on the value you want stored in the variable |
|  | Scope | A way to determine where the variable can be used. |
|  | Table of variable | An internal table kept by all programming languages at run time which holds the name of the variables, its type (if the language is static type) and the memory location where the variables value is stored |
|  | Reference ./ Passed by Reference | Using just the reference in the Table of Variable not the actual value held in the memory |