Unit II: Creating Code

Chapter 9: Variables in Alice 3

A **variable** is a named location in memory. In this lesson, you will learn about variables and how to use them.

In this chapter, you will learn about variables and how they are used to store information in an Alice program. You will learn how to declare variables, choose appropriate data types, and assign values to them. You will also explore how the values stored in variables can be changed and used throughout a program to control actions, perform calculations, and make your programs more flexible.

Objectives

- Name the different types in Alice.
- Name variables and objects correctly.
- Declare a variable and give it an initial value.
- Use the assign statement to give a variable a new value.
- Select and use functions to ask the user a question and store the result.
- Experiment with the tools available in Alice3.
- Use constants appropriately.

Type

There are four primitive types you can select in Alice 3:

- **Double** is a number with decimal places, for example 0.25 or 1.3
- **Integer** is whole numbers with no decimal places such as 4, -2 or 0.
- **Boolean** are things that can be **true** or **false**,
- **String** is a word such as "hello" or "goodbye".

Java Style Code

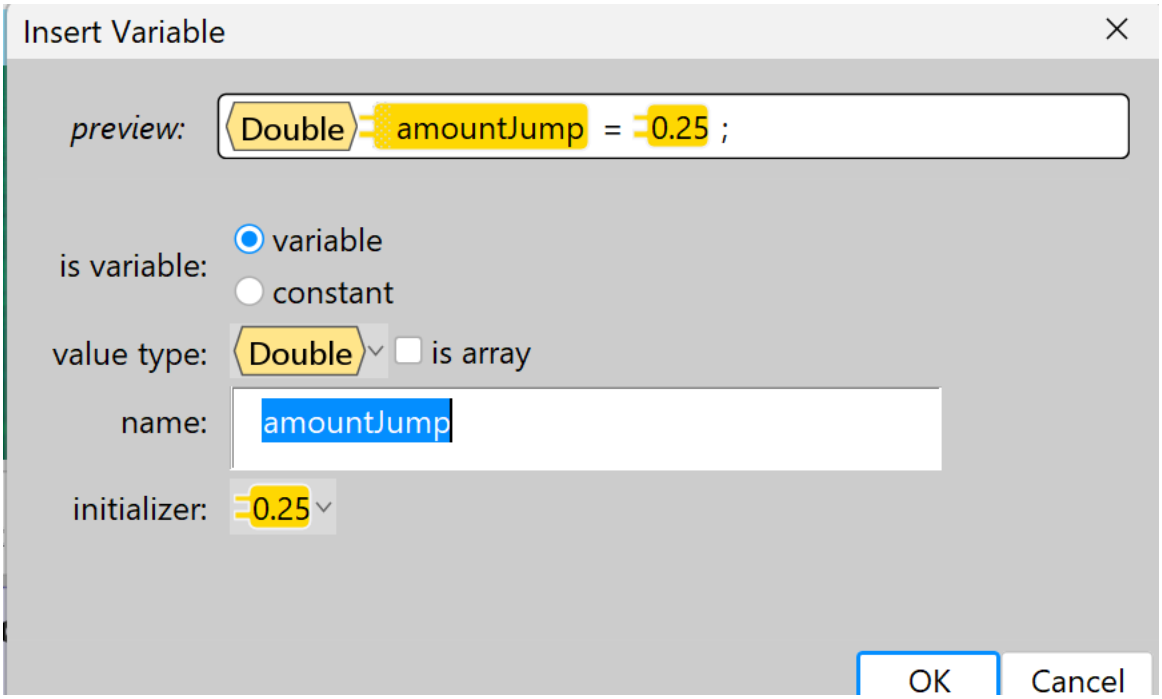
In Alice you can see the code either in Java style or Alice style. All the examples and lessons at zebra0.com use **Java style**. If you are using Java style you will see the types **Double**, **Integer**, **Boolean** and **String**. In the Alice style, you will see **DecimalNumber**, **WholeNumber**, **Boolean** and **TextString**. You change the preferences by selecting from the menu File, Preferences, Programming Language, Java. This change is important, because then your code will look like that shown in the lessons. Also, it is a good way to get used to what Java looks like, as that is a language you are likely to learn in the future.

Double

All the **move**, **turn**, and **roll** procedures require a **double** value as the argument. We would like to have a bunny hop. Jumping or hopping usually involves moving some amount up, then move the same amount down. This is often a trial-and-error process of figuring out what value for the movement looks right. Instead of trying a value, then having to change it in two places (for both the up and down movements) we will use a variable. Because this is a movement, that value will be a double.

Insert Variable

Drag the variable tile into the code. In the pop-up window, select Double as the type. Name it amountJump, and give it an initial value of 0.25 as shown below. The preview shows the statement that will be added to the code.



Variable Names

- A variable name must start with a letter of the alphabet or an underscore.
- After the first letter, there can be additional letters, digits and underscores.
- There cannot be any spaces in a variable name.
- Words that have a special meaning such as **Do** or **Boolean** should not be used to name variables.
- The name of the variable should indicate its purpose. Variable names like a, b and c are not very good names for variables unless you are using them for the three sides of a triangle.

Camel Case

If two words are joined together (you can't have any spaces), an uppercase letter for the second word can improve the clarity. Example: **amountJump**. (This style is referred to as **camel case**, but you can also use an underscore.)

Use the Variable

Next, we will select the bunny and drag the **move** command to the code. Select **up**, then look at the bottom of the pop-up window and select the variable **amountJump** instead of a type. You can do this again to move down or copy the first statement to the clipboard and then drag the clipboard into the code and change the direction to **down**.

The code will look like this:

```
public void myFirstMethod() {
    // Bunny Hop
    // Programmer : Janet Joy
    Double amountJump = 0.25;
    this.bunny.move( MoveDirection.UP, amountJump );
    this.bunny.move( MoveDirection.DOWN, amountJump );
 }
```

Run the program. If you think the bunny jumps too high, or not enough, change the value of the variable **amountJump**. Adjust the value until you like the animation.

Add another Double

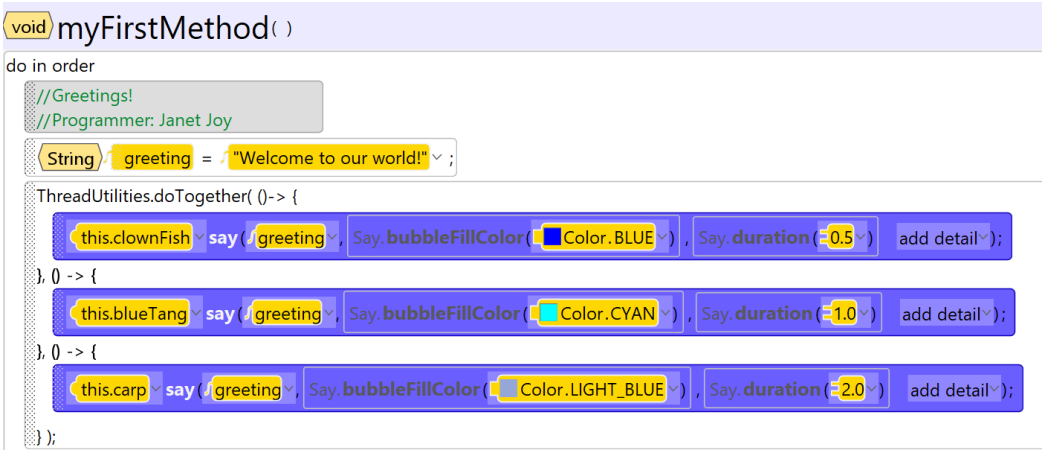
Next, we would like the bunny to move forward a bit at the same time he jumps up and down. We add another **Double** for the move amount. Add **doTogether** blocks so that he goes up and forward, then down and move a bit more forward.

```
public void myFirstMethod() {
    // Bunny Hop
    // Programmer : Janet Joy
    Double amountJump = 0.25;
    Double moveAmount = 1.0;
    doTogether( () -> {
        this.bunny.move( MoveDirection.UP, amountJump );
    }, () -> {
        this.bunny.move( MoveDirection.FORWARD, moveAmount );
    } );
    doTogether( () -> {
        this.bunny.move( MoveDirection.DOWN, amountJump );
    }, () -> {
        this.bunny.move( MoveDirection.FORWARD, moveAmount );
    } );
 }
```

When we run this, we see that the movement **up** is not enough and the **forward** movement is too much. *(Maybe we would also like to change the **duration** and **animationStyle** of the movement.)* However, by using a variable, we don't need to change the double values in two places each time we test it. This will make getting the bunny to hop much easier.

Strings

I have an ocean scene with three fish. I would like the fish to all say the same thing at the same time. I have declared a String variable named greeting and given it a value of "Welcome to our world." Note that even though all the fish say the same thing, we can change other arguments to the say procedure.



Assign

The assign statement lets the programmer give a new value to a variable at run time. In the code below, we copy the whole **doTogether** block to the clipboard. We will use the assign command to give a new value to greeting, then drag the clipboard into the code. Notice that the first doTogherBlock and the second doTogether block are identical, but now the fish say "Goodbye."

```
public void myFirstMethod() {
    // Greetings!
    // Programmer : Janet Joy
String greeting = "Welcome to our world!";
doTogether( () -> {
        this.clownFish.say( greeting, Say.bubbleFillColor( Color.BLUE ), Say.
duration( 0.5 ) );
    }, () -> {
        this.blueTang.say( greeting, Say.bubbleFillColor( Color.CYAN ), Say.d
uration( 1.0 ) );
    }, () -> {
        this.carp.say( greeting, Say.bubbleFillColor( Color.LIGHT_BLUE ), Say
.duration( 2.0 ) );
    } );
greeting = "Goodbye!";
```

```
doTogether( () -> {
        this.clownFish.say( greeting, Say.bubbleFillColor( Color.BLUE ), Say.
duration( 0.5 ) );
    }, () -> {
        this.blueTang.say( greeting, Say.bubbleFillColor( Color.CYAN ), Say.d
uration( 1.0 ) );
    }, () -> {
        this.carp.say( greeting, Say.bubbleFillColor( Color.LIGHT_BLUE ), Say
.duration( 2.0 ) );
    } );
// fish turn and dive down into the water...
 }
```

Experiment

Using variables, create the code to make the fish dive down into the water.

Variable Scope

Scope refers to where in the program a variable is available. Scope is usually limited to the control structure, such as **myFirstMethod** or **doTogether**, where the variable is declared. In the code below, name and greeting are declared in **myFirstMethod** and are available anywhere in **myFirstMethod**. When we run this the alien says “Hello, my name is Ivan.”

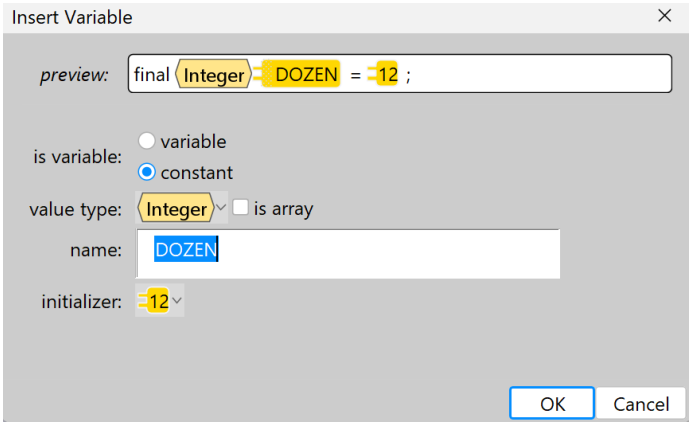
```
public void myFirstMethod() {
    // Alien and Variables.
    // Programmer : Janet Joy
    String name = "Ivan";
    String greeting = "Hello, my name is ";
    this.alien.say( greeting + name );
 }
```

In the next program, name is declared at the top level of my first method, but greeting is declared inside the together block. The statement where the alien says greeting + name will work inside the **doTogether** block, but he can only say his name outside the block where greeting was declared.

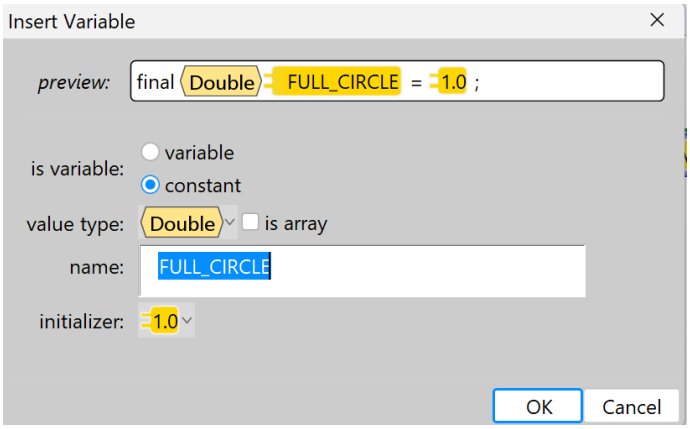
```
public void myFirstMethod() {
    // Alien and Variables.
    // Programmer : Janet Joy
    String name = "Ivan";
    doTogether( () -> {
        String greeting = "Hello, my name is ";
        }, () -> {
        this.alien.say( greeting + name );
    } );
    this.alien.say( name );
 }
```

Variable vs. Constants

When we insert a variable by dragging the variable tile into the code window, the pop-up box gives us a choice of **variable** or **constant**. A variable can **vary** or change its value. A constant never changes its value. Constants are usually declared using all upper-case letters. **PI** is a constant, it has a value of 3.14, and never changes. We could also declare a constant **DOZEN**, it has a value of 12 and never changes.



We have a scene with a dolphin and want him to do a backflip. We will declare a **constant** FULL_CIRCLE, it has a value of 1.0 and will never change. (You can edit it to change its value, but you cannot change it at run time.)



The code:

The code for the dolphin backflip is shown below:

```
public void myFirstMethod() {
    // Constants
    // Programmer : Janet Joy
    Double FULL_CIRCLE = 1.0;
    this.dolphin.turn( TurnDirection.BACKWARD, FULL_CIRCLE );
 }
```