

Unit II: Creating Code

Chapter 15: Calculations and Functions

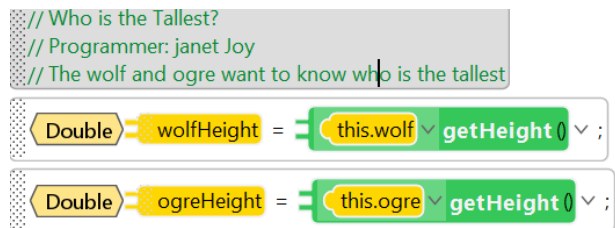
In this chapter, you will learn how to use built-in functions, such as **getHeight**, to retrieve information about objects in the Alice virtual world. You will use functions in calculations and learn how they can be used to create counters and determine totals. You will also learn how to create your own functions, allowing you to organize code, perform specific calculations, and return information that can be used elsewhere in an Alice program. You will also use arithmetic operators to move the characters to precise locations, to find totals, and other calculations.

Objectives

- Build arithmetic expressions using functions.
- Use variables, functions, and assign statements.
- Create code to do calculations using multiple variables and steps.
- Write a function to do calculations.
- Create a function to find the taller of two bipeds.

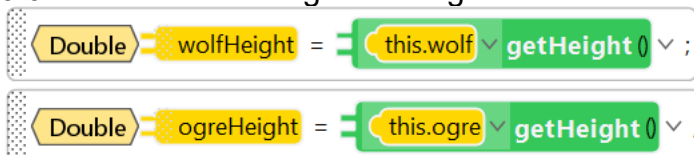
Find Tallest

The wolf and ogre want to know who is the tallest. Declare a **double** variable **wolfHeight** with an initial value of 0. *(It doesn't matter what value you initialize it to because we are going to replace it. You can use a 0 to indicate a temporary value.)*



Next, we will use a built-in function, **getHeight**, to replace the 0.0 with the wolf's actual height.

Select the **wolf**, then click the **Functions** tab (next to the procedures tab). Select the **getHeight** function and drag it to the variable declaration to replace the 0.0. Do the same thing for the ogre. Your variable statements should look like this:



We can't assume that if the wolf is not taller than the ogre, then the ogre is taller. There are three possibilities: the ogre is taller, the wolf is taller, or they are the same height. The code below checks for those three possibilities.

```

void myFirstMethod( )
do in order
  // Who is the Tallest?
  // Programmer: Janet Joy
  // The wolf and ogre want to know who is the tallest

  Double wolfHeight = this.wolf.getHeight();
  Double ogreHeight = this.ogre.getHeight();

  // They say their height to help us check the code.

  ThreadUtilities.doTogether( ()-> {
    this.wolf.say("My height is " + wolfHeight, Say.duration(1.0));
  }, ()-> {
    this.ogre.say("My height is " + ogreHeight, Say.duration(1.0));
  });

  // Find the tallest

  if ( ogreHeight > wolfHeight ) {
    this.ogre.say("I am the tallest!");
  } else {
    // We can't assume that if the ogre isn't taller than the wolf that the wolf is taller
    if ( ogreHeight == wolfHeight ) {
      this.ogre.say("We are the same height!");
    } else {
      this.wolf.say("I am the tallest!");
    }
  }
}

```

You can test the program by changing the heights in setup scene, or by using the set height procedure.

If we add another character, the program gets more complicated. An easy way to check for the **tallest** (or other minimum or maximum) value is to assume that the first value is the **tallestHeight**. Then compare each of the other values, one at a time, to see if it is taller. If it is set, **tallestHeight** to that new value. After testing each value, compare each character's height to the tallest Height, if so that character is one of the tallest (or the tallest.)

Add a monkeyKing to the previous program.

Add a variable **monkeyKingHeight** using the function **getHeight** as before.

The code is shown below:

```
public void myFirstMethod() {
    // Who is the Tallest?
    // Programmer : Janet Joy
    // The wolf, monkeyKing and ogre want to know who is the tallest
    Double wolfHeight = this.wolf.getHeight();
    this.wolf.say( "My height is " + wolfHeight, Say.duration( 10.0 ) );
    Double ogreHeight = this.ogre.getHeight();
    Double monkeyKingHeight = this.monkeyKing.getHeight();
    // Find the tallest
    // Step 1 : assume the first one is the tallest
    Double tallestHeight = wolfHeight;
    // Step 2 : Compare 2nd height to tallest height
    if( ogreHeight >= tallestHeight ) {
        tallestHeight = ogreHeight;
    } else {
        // No else is needed
    }
    // Step 3 : Compare 3rd to tallest height
    if( monkeyKingHeight >= tallestHeight ) {
        tallestHeight = monkeyKingHeight;
    } else {
        // No else is needed
    }
    this.wolf.say( "The tallest height is " + tallestHeight );
    doTogether( () -> {
        if( tallestHeight == monkeyKingHeight ) {
            this.monkeyKing.say( "I am one of the tallest." );
        } else {
            // No if needed
        }
    }, () -> {
        if( tallestHeight == ogreHeight ) {
            this.ogre.say( "I am one of the tallest." );
        } else {
            // No if needed
        }
    }, () -> {
        if( tallestHeight == wolfHeight ) {
            this.wolf.say( "I am one of the tallest." );
        } else {
            // No if needed
        }
    } );
}
```

Tip: Use the clipboard when creating this code!

Experiment: Add a fourth character. Change the heights in either setup scene or using the **setHeight** procedure.

Experiment: Count how many have a height equal to the **tallestHeight**. Change the wording to say either “I’m the tallest”, “I’m one of the tallest!” or “We are all the same height!”

Find Closest Place to Hide

A clown fish is swimming when suddenly a shark appears. The fish has three places to hide: a treasure chest, a cave, and some seaweed.

Which one is closest?

Because we want to find the closest object, not just the distance, we need a variable of the best gallery class that includes all the hiding places.



Drag a variable tile into the code, then for the class, select Gallery Class, then Prop. Use any one of the places as the initial value.

Prop closestPlace = this.cave ;

Gallery Class

Filtering

Assignable From Contains

Select class via the lowest common ancestor assignable from the items below:

- myScene
- ground
- camera
- clownFish
- cave
- treasureChest
- seaweed

Selection

- SScene
 - Scene
- STurnable
 - SMovableTurnable
 - SModel
 - SJointedModel
 - SSwimmer
 - Swimmer
 - Fish
 - ClownFish
 - SProp
 - Prop
 - Cave
 - TreasureChest
 - Seaweed
 - SCamera
 - SGround

Select a class

The method of finding the closest place is to assign one as the initial value, then compare each of the other places. If we find one that is closer, then we will assign that one as the closest place.

It will be easier to understand the code if we create variables for the distance from the fish to the hiding place. To start, we create a Double variable named **caveDistance** with an initial value of 0.0.

Double caveDistance = 0.0 ;

It doesn't matter what initial value we give it, because we are now going to replace that value with the function **getDistanceTo**:

Select the fish, then select the function tab, then drag **getDistanceTo** to replace the 0.0 in the variable declaration.

```
Double caveDistance = this.clownFish.getDistanceTo(this.cave);
```

We could repeat that same sequence for the treasureChest and seaweed, but it is easier to use the clipboard: Right click on the caveDistance statement, and select copy to clipboard. Now drag the clipboard into the code. Right click on the name **caveDistance** and select rename; Then change the cave to the treasure chest.

```
Double treasureChestDistance = this.clownFish.getDistanceTo(this.treasureChest);
```

```
void myFirstMethod( )
```

do in order

```
//Save Nemo!
//Programmer: Janet Joy
//Nemo is swimming when suddenly a shark appears.
//He has three places to hide: a treasure chest, a cave, and some seaweed.
//Which one is closest?
```

```
Prop closestPlace = this.cave;
```

```
Double caveDistance = this.clownFish.getDistanceTo(this.cave);
```

```
Double treasureChestDistance = this.clownFish.getDistanceTo(this.treasureChest);
```

```
Double seaweedDistance = this.clownFish.getDistanceTo(this.seaweed);
```

At this point we have the code as shown above.

Find Closest Place

We started with assigning the **cave** as the closest place. Now we just compare the distance to each other place to the distance to the current closest place.

```

if( seaweedDistance < closestDistance ) {
    //the seaweed is closer
    closestDistance = seaweedDistance;
    closestPlace = this.seaweed;
} else {
    //we don't need an else, we just go on to compare the treasure chest either way
}

```

We can copy that code to the clipboard, and use it to compare the treasure chest distance.

The complete code is shown below:

```

public void myFirstMethod() {
    // Save Nemo!
    // Programmer : Janet Joy
    // Nemo is swimming when suddenly a shark appears.
    // He has three places to hide : a treasure chest, a cave, and some seaweed.
    // Which one is closest?
    Prop closestPlace = this.cave;
    Double caveDistance = this.clownFish.getDistanceTo( this.cave );
    Double treasureChestDistance = this.clownFish.getDistanceTo( this.treasureChest );
    Double seaweedDistance = this.clownFish.getDistanceTo( this.seaweed );
    Double closestDistance = caveDistance;
    if( seaweedDistance < closestDistance ) {
        // the seaweed is closer
        closestDistance = seaweedDistance;
        closestPlace = this.seaweed;
    } else {
        // we don't need an else, we just go on to compare the treasure chest either way
    }
    if( treasureChestDistance < closestDistance ) {
        // the seaweed is closer
        closestDistance = treasureChestDistance;
        closestPlace = this.treasureChest;
    } else {
        // we don't need an else, we just go on to compare the treasure chest either way
    }
    this.clownFish.turnToFace( closestPlace, TurnToFace.duration( 0.25 ) );
    this.clownFish.moveAndOrientTo( closestPlace, MoveAndOrientTo.duration( 0.5 ) );
}

```

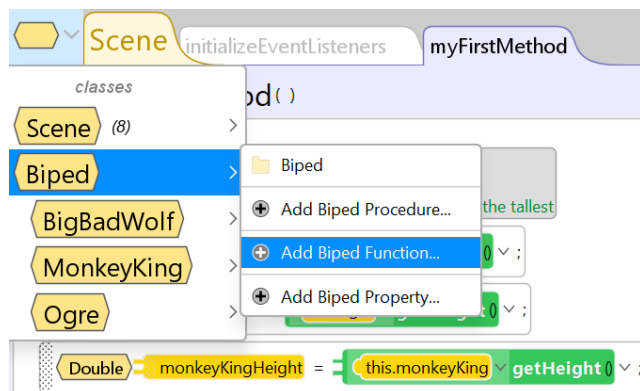
After completing the code, you should test it. If you want, you can add code for the fish to say the distance to each location. That will help you determine if the code is correct. Next go to setup scene and move the objects around a bit and see if it still finds the closest place. What if two locations are the same distance from the fish? Will it still find one that the fish can swim to?

Experiment: Add a fourth location.

Experiment: What if the shark is near the cave? How would you find a safe place to hide?

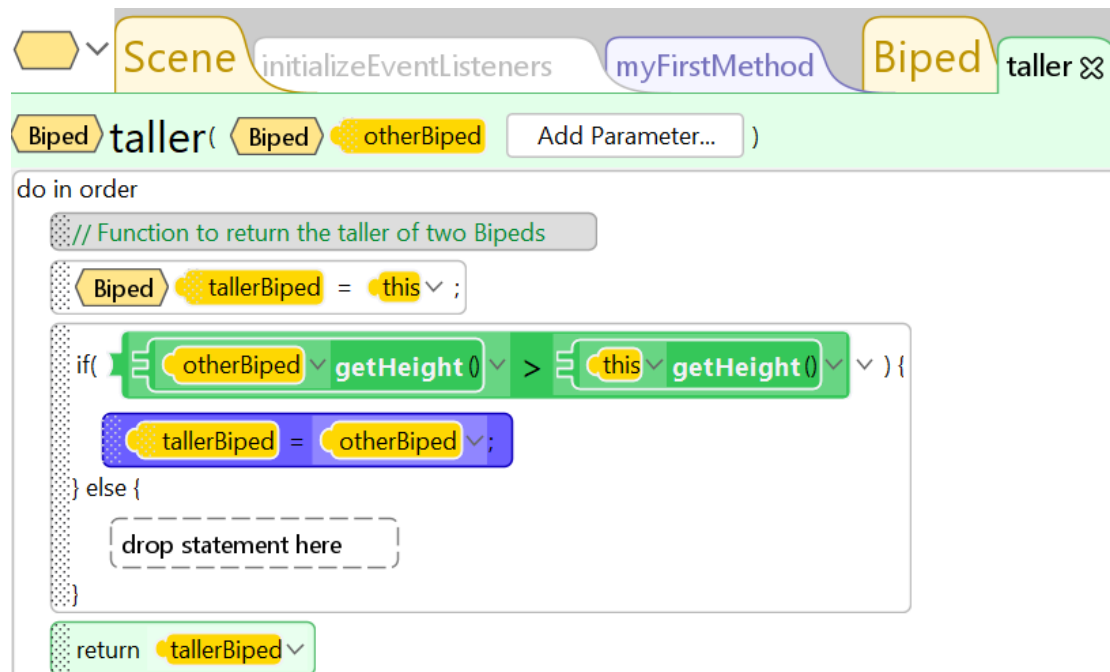
Taller Function

We are going to write a function to find the taller of two bipeds. We will write a function the same way we write a procedure. The difference between a procedure and a function is that a procedure performs an action, a function returns a value. So a function will need to specify a type. Click the 6 sided shape at the top of the code window, then select Biped, + Add Biped Function.



Select Gallery Class as the type, then Biped: Name the function **taller**.

Build the code as shown below:



Step 1: Add a parameter that is Gallery Class, Biped. Name the parameter **otherBiped**.

Note that in this function, there are two Bipeds: **this** and **otherBiped**.

Step 2: Add a **variable** that is also Gallery Class, Biped. Name it **tallerBiped** with an initial value of **this**.

Step 3: Build the **if** statement. It is comparing two **double** values: the height of this and the height of otherBiped. initially it is **if(0.0 > 0.0)** But then we replace the 0.0 with the **getHeight** functions for **this** and **otherBiped**. (You can select them in the window on the top left.)

```
void myFirstMethod( )
```

do in order

```
// Who is the Tallest?  
// Programmer: Janet Joy  
// The wolf, monkeyKing and ogre want to know who is the tallest
```

```
Double wolfHeight = this.wolf.getHeight();
```

```
Double ogreHeight = this.ogre.getHeight();
```

```
Double monkeyKingHeight = this.monkeyKing.getHeight();
```

```
// Show the heights
```

```
ThreadUtilities.doTogether( ()-> {
```

```
    this.wolf.say( "My height is " + wolfHeight + addDetail());
```

```
}, ()-> {
```

```
    this.ogre.say( "My height is " + ogreHeight + addDetail());
```

```
}, ()-> {
```

```
    this.monkeyKing.say( "My height is " + monkeyKingHeight + addDetail());
```

```
});
```

```
// Find the tallest
```

```
// Step 1: biped is the taller of wolf and ogre
```

```
Biped biped = this.wolf.taller( this.ogre );
```

```
biped.say( "I am the taller one of wolf and ogre" + addDetail());
```

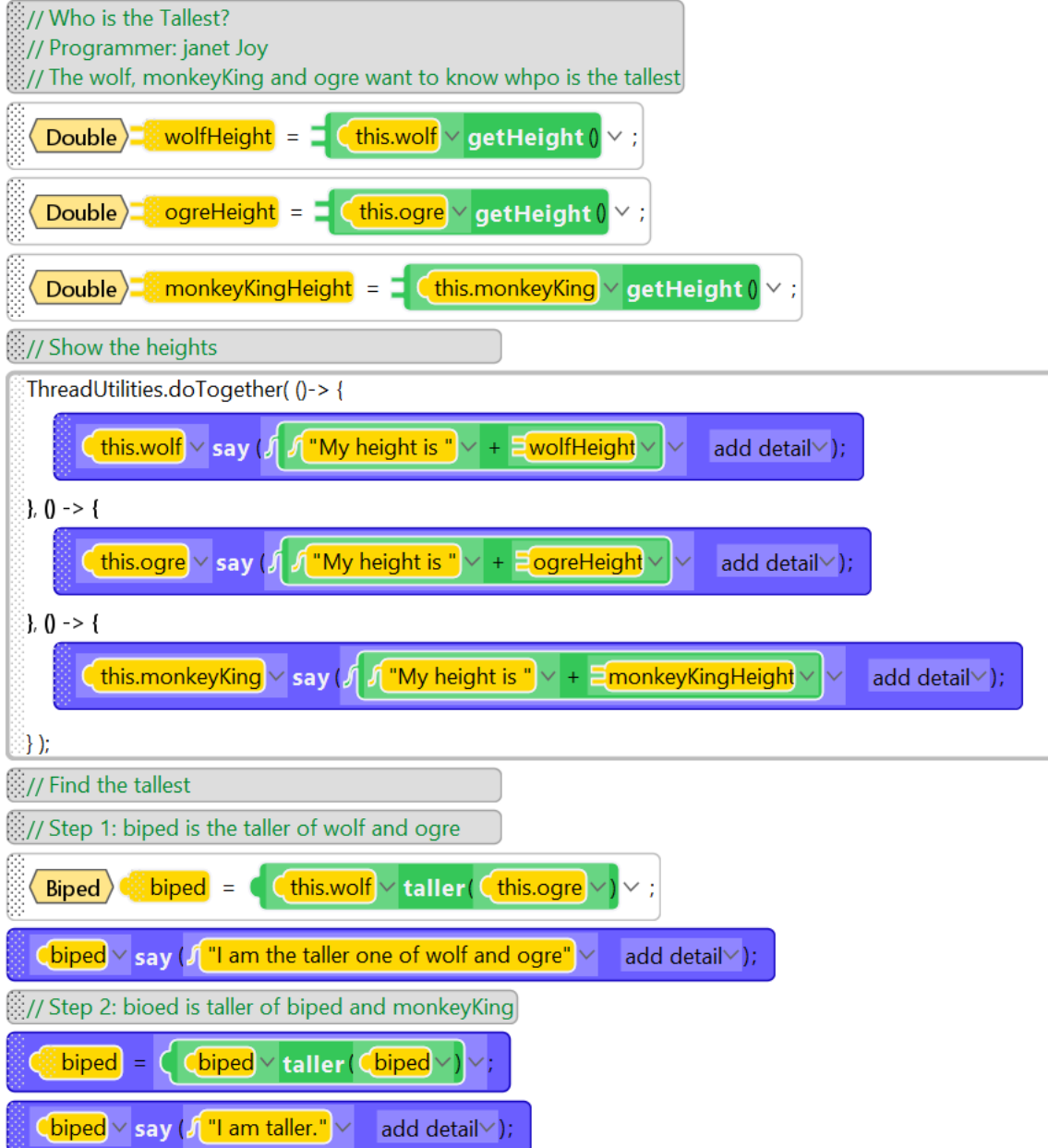
```
// Step 2: biped is taller of biped and monkeyKing
```

```
biped = biped.taller( this.monkeyKing );
```

```
biped.say( "I am taller." + addDetail());
```

void myFirstMethod()

do in order



Experiment: Add another Biped.