

UNIT 3. VARIABLES AND INTEGERS

Summary

In this section, you will...

- ♦ Learn how computers can store small bits of information as variables.
- ♦ Make your own variables, store information in them, and get information out.
- ♦ Get and install the Processing program which enables you to write Processing computer programs.
- ♦ Learn the details of syntax surrounding variables.
- ♦ Learn about integers.
- ♦ See how computers do the basic integer operations of addition, subtraction, multiplication, and division – and see the issues with integer division.
- ♦ See how computers use the remainder operation (modulo) with integer division.
- ♦ Learn about basic debugging by printing out the contents of variables

Learning Objectives

After finishing this unit, you will be able to write basic Processing programs that use simple integer variables. You will be able to:

- ♦ Create a new integer variable and give it a unique name.
- ♦ Store information into a variable (e.g., the number 10).
- ♦ Get information out of a variable.
- ♦ Create a variable and give it an initial value in one line of code.
- ♦ Create multiple variables in one line of code.
- ♦ Add, subtract, multiply, and divide integers.
- ♦ Get the remainder when dividing two integers.
- ♦ Apply order of operations to figure out the result of a calculation.
- ♦ Use the built-in `println` command to look inside variables for investigating your errors.

How to Proceed

- ♦ Read the unit content.
- ♦ Have a Processing window open while you read, to follow along with the examples.
- ♦ Do the sets of exercises in the **Check your Understanding** sections.
- ♦ Re-check the **Learning Objectives** once done.



3.1 Introduction

What is a variable? We use them all the time in algebra – there, we use a letter or a Greek symbol to represent some number. Some common variables in math are x , y , k , and l . Variables are similar in programming, although we use them less for algebra and proofs (yay!). Basically, it is a way to store information. Computers use this all the time. For example, when you log into a website, the website stores your name and personal information in variables, and then uses that information in its processing.

*Hi [name], welcome to our site! We see that you come from [city].
People in [city] previously bought the following items from us...*

Variables help the programmer write the template and the general case, and fill in the blanks later.

Look at the following program:

```
/******  
* Cat Face! Draw a cat face on the screen  
* author: Teo the dog  
* version: try #awesome  
* purpose: to show how a cat can be drawn  
*****/  
  
size(500,500); // make a 500x500 canvas  
  
//draw the head  
ellipse(250,250,300,300);  
  
//draw the ears  
triangle(375,80,300,150,400,200);  
triangle(125,80,200,150,100,200);  
  
//draw the eyes  
ellipse(175,225,60,30); // left eye  
ellipse(175,225,15,30);  
ellipse(325,225,60,30); // right eye  
ellipse(325,225,15,30);  
  
//whiskers!  
line(250,300,200,275);  
line(250,300,300,275);
```

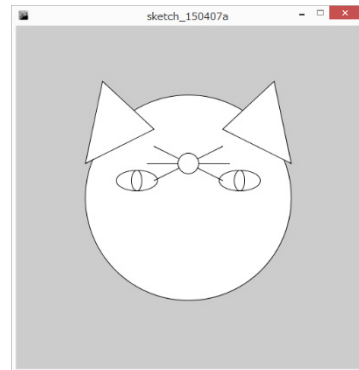
```

line(250,300,190,300);
line(250,300,310,300);
line(250,300,200,325);
line(250,300,300,325);

// draw the nose. draw after whiskers for nice overlap effect
ellipse(250,300,30,30);

```

Here is our cat program again. Now, what if I wanted to move the whiskers and nose to be a little higher? Let's try 100 pixels higher. To do this we need to change all the *y* coordinates of the whiskers and nose – a real pain in the donkey. Go and do it and press play. You will find that my approximation was off – 100 pixels is way too high and now the cat has a nose on the forehead. So we need to go back and manually do it again, trying a new *y* offset and changing all those values before seeing the result. Wouldn't it be nice if there was a better way?? Well, variables help solve this problem. Remember in algebra we can do things like this:



```

x = 25 // store the # 25 in x
y = 2x // grab value from x (25), multiply it by 2 (=50),
        store in y
z = 4x + 7 // grab value from x (25), multiply by 4 (=100),
            add 7 (=107), store result in z

```

y and *z* are defined relative to *x*. If instead *x* was 30, then *y* and *z* are calculated differently, too. It would be nice if we could rewrite our program in a similar way:

```

noseCenterY = 200

draw straight-out whisker at noseCenterY

... (do this for all the drawing commands)

```

Then, if we want to move the nose, we just change the value stored in our variable, and everything is fixed! All of these are called variables because what is stored in them can change all the time.

3.2 Variables as Boxes

You can think of variables as boxes that store information. Imagine you are organizing your office or basement (see the picture). Now, in this imaginary scenario, you are very strict on how you must organize things, so you collect a bunch of boxes



and label them with what goes in them. These labels may be things like books, research papers, and toys. AND, you will be VERY

And no exceptions! NEVER put a research paper in a toy box



upset if someone mixes things around. No toys in the book box. If this was processing, those boxes are variables, and just like in our scenario, processing is very strict about what goes in each box. When you create a

variable (create a box), you need to tell processing about what kind of data you are going to put into it, for example, a number, or some text, or some music! These are called *data types*.

3.3 Your First Data Type: the Integer

What is an integer? (high school math!!). If you remember, it is simply a whole number with no fractional part. The following are integers: 0, 42, -99, and 200. The following are not integers: 3.14, 99.1.

What is the biggest integer? Theoretically, there is none. However, the larger your integer is, the more memory a computer needs to store it, so unfortunately, we cannot have infinitely large integers in Processing since computers do not have infinite memory. In fact, in Processing integers have a pre-defined limit on the range of numbers that can be stored in it

the range -2,147,483,648...2,147,483,647

That is over a 4 billion number range. Clearly, these exact numbers were chosen for obvious reasons. That was sarcasm. Don't bother memorizing these numbers, just have a general sense – about minus 2 billion to about 2 billion.

If you need larger or smaller numbers, there are options, but we'll talk about them later.

It's so obvious! Processing uses 32 bits, which has 2^{32} possibilities = 4,294,967,296. If you divide that into positive and negative, divide by two, you get 2,147,483,648. The positive range looks to be one less since 0 is included as a non-negative number. Clearly.



WHAT?? NO FRACTIONS OR DECIMALS??

That's right – computers are way faster with whole numbers, so we generally resort to those. There are also problems when fractions and whole numbers mix which

comes up in a later unit. We'll learn about what to do when we need decimals later.

3.4 Variables in Processing: Syntax

To work with variables in processing, we need to solve three problems:

- ♦ how to create a variable
- ♦ how to put data into a variable
- ♦ how to get data back out of a variable

here is the syntax to create a variable in Processing:

```
variableType variableName;
```

So far, we only know one type, the integer. In Processing, this is converted to the shorthand `int`. So, to create an integer variable called `noseCenterX` to store the `x` coordinate of the cat's nose:

```
int noseCenterX;
```

don't forget the semi colon!

NOTE: Variables must be created (declared) before you use them. (remember that the program runs top to bottom). To avoid this mistake altogether, generally, we put these at the top of our program.

The next problem is how to put data into the variable. The syntax for this is:

```
variableName = data;
```

So in our example, since the cat's nose is centered at the `x` coordinate of 250:

```
noseCenterX = 250;
```

Note: the hand-written data like an actual number, or some text, is called "literals" in programming speak.

So now we can create variables, and we can store data into them, how do we look at them? This is very simple. You just use the variable name anywhere you can use raw data. For example, we can now rewrite the following command:

```
// draw the nose. draw after whiskers for nice overlap effect  
ellipse(250,200,30,30);
```

as

```
// draw the nose. draw after whiskers for nice overlap effect
ellipse(noseCenterX,200,30,30);
```

Let's also add a variable for `noseCenterY`, and update all of the drawing to use our new variables. You end up with the following code:

```
/******
 * Cat Face! Draw a cat face on the screen
 * author: Teo the dog
 * version: try #awesome
 * purpose: to show how a cat can be drawn
 *****/
int noseCenterX;
noseCenterX = 250;
int noseCenterY;
noseCenterY = 300;

size(500,500); // make a 500x500 canvas

//draw the head
ellipse(250,250,300,300);

//draw the ears
triangle(375,80,300,150,400,200);
triangle(125,80,200,150,100,200);

//draw the eyes
ellipse(175,225,60,30); // left eye
ellipse(175,225,15,30);
ellipse(325,225,60,30); // right eye
ellipse(325,225,15,30);

//whiskers!
line(noseCenterX,noseCenterY,200,275);
line(noseCenterX,noseCenterY,300,275);
line(noseCenterX,noseCenterY,190,300);
line(noseCenterX,noseCenterY,310,300);
```

```
line(noseCenterX,noseCenterY,200,325);
line(noseCenterX,noseCenterY,300,325);

// draw the nose. draw after whiskers for nice overlap effect
ellipse(noseCenterX,noseCenterY,30,30);
```

Now, the nose center drawing uses the information in our variable instead of a hard-coded *literal* number. If you change the `noseCenter` variables around, all the nose commands move nicely. However, there is a problem here!! The whisker ends do not follow along, since we only have the line *starting points* rely on the variable, and not the end points. We'll have to fix this. However, let's learn a few more things about variables first.

3.5 Additional Variable Details

Here is a final list of important points regarding the use of variables. First, when you create a variable, you often want to assign it a value right away. Because of this, you'll often see code like the following:

```
int noseCenterX;
noseCenterX = 250;
```

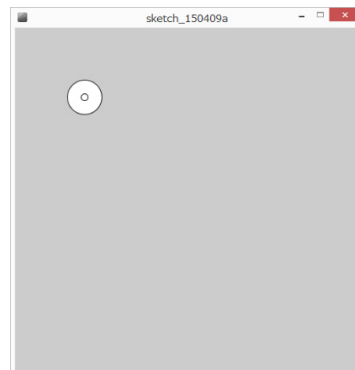
Programmers like shortcuts, so **there is a clever shortcut for this special case, it lets you create a variable (declare it) and give it a value (assign to it) in one go:**

```
int noseCenterX = 250;
```

With this trick, you can turn two lines of code into one!

Variables are variable: **you can change variables at any time!!** And remember that the program runs top to bottom. Check out the following code

```
int circleSize = 50;
ellipse(100,100,circleSize,circleSize);
circleSize = 10;
ellipse(100,100,circleSize,circleSize);
// same as above
```




Even though we type the exact same ellipse command twice, the second one has a different result because the value of `circleSize` changed. Let's quickly step

through what happens, going from top to bottom.

1. A new variable is created called `circleSize`, and the number 50 is stored in there.
2. An ellipse is drawn at (100,100) of size (`circleSize`, `circleSize`). Since `circleSize` currently has the value 50, it is drawn at size (50,50).
3. The number 10 is stored in `circleSize`. The old value, 50, is thrown away.
4. An ellipse is drawn at (100,100) of size (`circleSize`, `circleSize`). Even though this is the same command as in line 2, the result is different because the variable has new data. Now, it is drawn size (10,10).

Further, once a variable has been declared, then you cannot declare it again. Processing gets very confused if you do this! Check out the following code:


```
int circleSize = 50;
ellipse(100,100,circleSize,circleSize);
int circleSize = 10;
ellipse(100,100,circleSize,circleSize); // same as above
```



See the difference with the prior example? Toss it into Processing and see what happens. When you click run, you unfortunately get an error: Duplicate local variable `circleSize`. **You can't have two variables with the same name.** To fix this, you need to remove the `int` in the second `circleSize`, so that you are not trying to create a new one, but instead, just copy a new number to the existing variable. This is a very important distinction – in the first (not-working case) we are asking processing to make a new variable with the same name. In the second case, we are re-using an existing variable and giving it new data (throwing the old data away).

What happens if you create a variable, never put anything into it, but try to use it? For example

```
size(500,500);
int circleSize;
ellipse(100,100,circleSize,circleSize);
```



If you try to run this, Processing will say: The Local variable `circleSize` may not have been initialized. and it won't run. This makes sense, since what would you expect the circle size to be if you never set a value? **Variables must be initialized – given a value – before they are used.**

You cannot just name a variable anything, there are rules. However, you have a lot of flexibility for variable names. Generally you should try to pick something nice and descriptive to help you read your code more clearly. Here are the restrictions:

- ♦ No spaces! You cannot do: `int circle size;` as it confuses Processing.
- ♦ No special characters `!"#%&'()-=^[]{} –` the exception is that underscore is allowed: `_` (some people use it as space) and `$` is allowed, although no one uses it since it looks funny (e.g., `int a$$e$$e$`);
- ♦ Cannot start with a number, but can contain one
 - `int 4peace;` // cannot do this!
 - `int piece4;` // this is okay

Another problem is that some words in Processing already have meaning. These are called `reserved words`. You cannot use these as variable names because they are already in use. For example, you cannot make the following variable:

```
int int;
```

Processing will say: unexpected token: int. This is because it's trying to add meaning to your variable name and not see it just as a name. Processing also has some of its own variables kicking around, and you cannot use those names either.

You can find whole lists of reserved words and existing variables, but it's probably not worth your bother to memorize them. Instead, just be aware of the problem in case you stumble across a strange error when making a new variable. Try a similar name and see if the problem goes away, as you may have a conflict.

If you have a number of variables to declare of the same type, **you can actually declare multiple variables at once.** Here is the syntax:

```
type variable1, variable2, ...;
```

for example

```
int age1, age2, age3;
```

This is equivalent to

```
int age1;
int age2;
int age3;
```

except that it takes less space.

In addition, you can add in assignments here:

```
int age1 = 17, age2 = 32, age3 = 23;
```



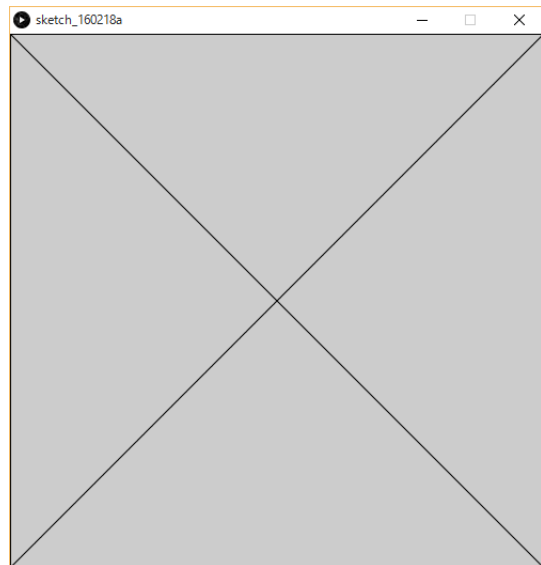
Check your Understanding



3.6 Check your Understanding: Variable Exercises

Exercise 1. Create four integer variables that represent the edges of the screen: `left` and `right` give the x coordinate of the left and right edge, and `top` and `bottom` give the y coordinates of the top and bottom edges.

- Create the 4 variables, one per line.
- Set your canvas to size 500,500 and give your variables their initial values. Hint: the right and bottom edges are *not* at 500.
- Use your variables and the line command to draw two lines to form an X: from the top left to the bottom right, and from the top right to the bottom left.
- Use your variables and the line command to draw a box around the edge of the screen.
- Re-write the above program to have all your variables declared and instantiated in a single command.



Exercise 2. The rules say that two variables cannot have the same name, but why does the following code actually work, with multiple variables given the same name?

```
int center = 10;
int radius = 10;
ellipse(center,center,radius,radius);
int Center = 20;
int Radius = 20;
ellipse(Center,Center,Radius,Radius);
```



Exercise 3. Update the cat face example from Section 3.4. Make sure to update your drawing commands to use your new variables as you make them.

- Add variables for the left eye and right eye (each eye needs two variables) to represent their center locations.
- Add variables to represent the pupil and eye width and height.
- Try moving the eyes around and changing their sizes using these variables to make sure you did it correctly.



Exercise 4. Try to do the following without typing them into Processing, as it will immediately tell you the answer. Which of the following variable declaration statements are invalid?

- `int cat;`
- `Int dog;`
- `int mi$$i$$ippi;`
- `int dog1, dog2, dog3;`
- `int 1mi$$i$$ippi, 2mi$$i$$ippi;`
- `int dog, dog;`
- `int MEANING_OF_LIFE = 42;`
- `int WHY?? = 15;`
- `int __SYSTEM_ERR;`
- `int &data_;`
- `int __data__ = -200;`

3.7 Integer Operations: addition and subtraction

Luckily, integer operations is a pretty simple topic. This will help us solve our cat whisker problem (above in section 3.4).

Let's start by looking at one of the previous examples, the cat picture. We had the following code for drawing a cat whisker

```
line(250, 300, 300, 325);
```

which we improved by adding named variables to it, as follows:

```
line(noseCenterX, noseCenterY, 300, 325);
```

Although we can nicely define the center of the nose (the starting point of the whiskers), the ends of the whiskers are still fixed numbers. If you remember, this made the end points of the whiskers stick on the face if we moved the nose.

If we look at the first case above (without the variable name), we can see that the end point of that particular whisker was 50 pixels to the right (250x start, 300x end),

and 25 pixels below (300_y start, 325_y end) the starting point. Once we set the starting point as a variable, we can use integer operations to calculate the other ones.

You can add two numbers together just by putting a plus sign between them. For example, you can do

I could have told you that...

```
int whiskerEndX;  
whiskerEndX = noseCenterX + 50;
```



now `whiskerEndX` will equal to 50 more than `noseCenterX`. In this case, 300.

You can use integer operations anywhere. Here, you don't need a new variable for the whiskers, we can just do the addition right inside the command:

```
line(noseCenterX, noseCenterY, noseCenterX+50,  
     noseCenterY+25);
```

NOTE: see how the command here broke across lines because it didn't fit on one? That is okay in processing.

To re-cap, we are drawing a line from `noseCenterX` and `Y`, to 50 pixels more than `X` and 25 pixels more than `Y`. In this case, processing looks inside the variables, gets the values, adds the literal numbers to that value, and uses the result in our line command. There are a lot of steps happening here, but you'll get used to the idea very quickly.

Now, if you change the `noseCenter` variables, that whisker will move along with the rest of the face, since both end points are calculated based on the whisker values.

Let's look at another whisker

```
line(250, 300, 200, 275);
```

In this case, the whisker ends 50 pixels to the left (250_x start, 200_x end), and 25 pixels above (300_y start, 275_y end) the start point. We cannot do this with addition, so we need subtraction.

Subtraction in Processing is just as simple as addition. Just use the minus sign: `-`. So, we can rewrite this operation as

```
line(noseCenterX, noseCenterY, noseCenterX-50,  
     noseCenterY-25);
```

If we now update all the whiskers this way, we can move the whole nose around

nicely simply by changing the values of the `noseCenterX` and `noseCenterY` variables.

If we look at our cat code, we can see another opportunity to use variables. This was included as one of the examples above. As a reminder, here are the eyes:

```
//draw the eyes
ellipse(175,225,60,30); // left eye
ellipse(175,225,15,30);
ellipse(325,225,60,30); // right eye
ellipse(325,225,15,30);
```

Notice how the narrow width of the pupil is 15, the height of the pupil is double that, which is the same as the height of the eye? Also, the width of the eye is 4 times the pupil width. These ratios make the eye look cool, and the pieces just touch nicely. If we were to set the pupil width as a variable, we would need multiplication to calculate the other sizes. For example, if the pupil width was 15, then the height is double the width, and the eye width is four times the pupil width. In our example, we can create a new variable and set it to our pupil width.

```
int pupilWidth = 15;
```

and then plop it into our code.

```
//draw the eyes
ellipse(175,225,60,30); // left eye
ellipse(175,225,pupilWidth,30);
ellipse(325,225,60,30); // right eye
ellipse(325,225,pupilWidth,30);
```

But now to go further we need multiplication.

3.8 Integer operations: Multiplication and Division

Multiplication in processing uses the asterisk – the `*` symbol (shift-8 on north American keyboards). We can use the multiplication operator on our `pupilWidth` variable to calculate the remaining widths and heights:

```
//draw the eyes
ellipse(175,225,pupilWidth*4,pupilWidth*2); // left eye
ellipse(175,225,pupilWidth,pupilWidth*2);
ellipse(325,225,pupilWidth*4,pupilWidth*2); // right eye
ellipse(325,225,pupilWidth,pupilWidth*2);
```

As you can see, we just multiply the width by 2 and 4 to get our desired width and heights.

You could imagine that this whole operation could be reversed, and calculated with respect to the eye width (the biggest number) instead of the pupil width (the smallest number). The eye height is half the eye width. The pupil height is also half the eye width. The pupil width is a quarter of the eye width. Do to this, we need division.

Division in Processing is accomplished with the / operator. You can envision that this is the symbol used in fractions, like $\frac{1}{2}$. I won't belabor the point, but the above example can be rewritten using division as follows.

```
int eyeWidth = 60;
ellipse(175,225,eyeWidth,eyeWidth/2); // left eye
ellipse(175,225,eyeWidth/4,eyeWidth/2);
ellipse(325,225,eyeWidth,eyeWidth/2); // right eye
ellipse(325,225,eyeWidth/4,eyeWidth/2);
```

Pretty straightforward. Unfortunately, division with integers in Java and Processing is not so simple, and is actually a problem that adds a whole bunch of confusion. Luckily, I'll get into that in a bit and for now, you can use it in this example. But be warned, there are huge caveats.

Go back to the cat example, and try to add more variables.

- ♦ Add variables for the center of the cats head. `headCenterX` and `headCenterY`
- ♦ Update the head ellipse to use it.
- ♦ Update the eye locations to use it.
- ♦ Update the ear locations to use it.
- ♦ Update the nose center to use it.

Now, if you do all this work, you have a cat head that you can move around the canvas JUST by changing the center variables. I strongly recommend you try it yourself. Here is my complete solution:

```
/******
 * Cat Face! Draw a cat face on the screen
 * author: Teo the dog
 * version: try #awesome
 * purpose: to show how a cat can be drawn
 *****/

// variables
int headCenterX = 250;
```

```

int headCenterY = 250;
int noseSize = 30;
int pupilWidth = 15;
int noseCenterX = headCenterX;
int noseCenterY = headCenterY+50;

size(500,500); // make a 500x500 canvas

//draw the head
ellipse(headCenterX,headCenterY,300,300);

//draw the ears
triangle(headCenterX+125,headCenterY-170,
         headCenterX+50,headCenterY-100,
         headCenterX+150,headCenterY-50);
triangle(headCenterX-125,headCenterY-170,
         headCenterX-50,headCenterY-100,
         headCenterX-150,headCenterY-50);

//draw the eyes
ellipse(headCenterX-75,headCenterY-25,
        pupilWidth*4,pupilWidth*2); // left eye
ellipse(headCenterX-75,headCenterY-25,
        pupilWidth,pupilWidth*2);
ellipse(headCenterX+75,headCenterY-25,
        pupilWidth*4,pupilWidth*2); // right eye
ellipse(headCenterX+75,headCenterY-25,
        pupilWidth,pupilWidth*2);

//whiskers!
line(noseCenterX,noseCenterY,noseCenterX-50,noseCenterY-25);
line(noseCenterX,noseCenterY,noseCenterX+50,noseCenterY-25);
line(noseCenterX,noseCenterY,noseCenterX-60,noseCenterY);
line(noseCenterX,noseCenterY,noseCenterX+60,noseCenterY);
line(noseCenterX,noseCenterY,noseCenterX-50,noseCenterY+25);
line(noseCenterX,noseCenterY,noseCenterX+50,noseCenterY+25);

// draw the nose. draw after whiskers for nice overlap effect
ellipse(noseCenterX,noseCenterY,noseSize,noseSize);

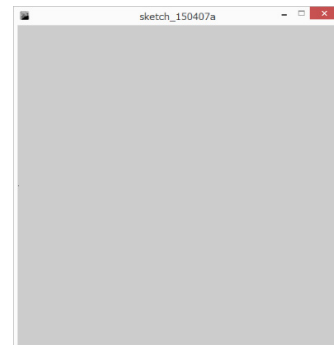
```

3.9 Advanced Integer Division and Modulo

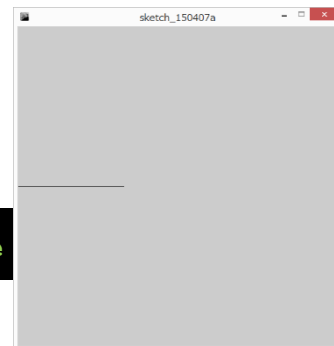
Try the following example. Make a program that draws a line across the screen, half way down. Make the line go some percentage across the screen. Set the percentage in a variable, and then calculate how far across to go. The code will look something like this

```
int percent = 33;
int targetX = percent/100*500; // /100 to make percent
line(0,250,targetX,250);
```

type this into Processing, with a 500x500 canvas, and you get the following output: huh? Where's the line? It seems to have not drawn. Let's try doing the calculating by hand: $33/100*500 = 165$. If you type 165 into the line code (replace the `targetX` variable with 165), you get the second diagram, with a line one third ways across the screen. Why did the line show up? Why does it work if we calculate it by hand, but not if we ask the computer to do it? They should be the same!



Finally! It's time for an amazing helper tool! Tada! **Processing has a way for you to peek at data!** Do you remember the console at the bottom of the Processing window? There is a processing command called `print` that lets you put data to that screen:



```
println(data); // prints out to the console
```

Let's try it. Modify your above program as follows:

```
int percent = 33;
int targetX = percent/100*500; // /100 to make percent
line(0,250,targetX,250);
println(targetX);
```

Now, in addition to drawing a line, this program will print the value of the `targetX` variable to the console, enabling you to take a peek at what is going on.

It shows the source of the problem – the `println` tells us that `targetX` is actually set to 0, not to our expected 165. Let's dig a little deeper, try the following command:

```
print(percent/100);
```

This is not how math works.. my head hurts...



Aha! You get 0 again! What did you expect? 0.33?

This is because we are working with integers, and not real numbers. Integer division does not work like you may think! In fact, it works how division worked way back in elementary school. Remember long division? If we did $33/100$ in long division, what do we get?

$$\begin{array}{r} 0 \\ 100 \overline{) 33} \\ \underline{0} \\ 0R33 \end{array}$$

0 remainder 33! Before we learned how to do decimals in school, we had remainders.

In computers, **Integer division always gives you the answer assuming you want the remainder style, not a fraction result.** What is $1/2$? 0 remainder 1. What is $11/3$? 3 remainder 2. What is $100/26$? 3 remainder 22.

This may seem very strange to you, but as you will see through this course, integer division has a lot of very useful applications. We will learn later how to do the decimal style with real numbers, as you may naturally expect. In all these cases, the remainder is thrown away, and the basic result (0, 3, 3, etc.) are kept. So how do we get the remainder? We use the modulo operator.

Modulo is a very unusual operation that you will learn a lot about in mathematics classes. Basically, it gives you the remainder when two numbers are divided. Let's look at one of the above examples, $11/3$. Remember, $11/3$ is 3 remainder 2.

```
println(11/3); // 3 is output
```

To get the remainder, we replace the $/$ symbol with $\%$, the percent sign.

```
println(11%3) // the number 2 is output, the remainder
```

So if $11/3$ is 3 remainder 2, we get the 3 with division, and we get the 2 (remainder) with modulo. You should practice this. What is the remainder when you do $10/2$? $5/2$? $11/3$? We will end up using remainder a lot later. For now, there is very little we can do with it that is fun and exciting, so we'll come back to it.

3.10 Order of Operations

Processing math operations follow standard order of operations from mathematics. What is the result of the following command?

```
println(3+2*6/3%4);
```

The answer depends on the order that you do the math. In high school, you learned a particular order that math happens – luckily, this follows here. Multiplication (and division and remainder) happens first, and then addition and subtraction:

- ♦ $3+2*6/3\%4$
- ♦ $\rightarrow 3+12/3\%4$
- ♦ $\rightarrow 3+4\%4$
- ♦ $\rightarrow 3+0$
- ♦ $\rightarrow 3$

If you do the addition first, you get the wrong answer.

Sometimes, the order of operations can be quite unclear. In this case, you can always use brackets to be sure and to enforce what you mean – just like in high school math:

```
println( 3+ (2*6/3)%4 );
```

The code inside the brackets happen first.



Check your Understanding



3.11 Check Your Understanding: Exercises

Exercise 1. In the following code, what does b equal at the end? 20 or 10? Hint – read top to bottom, and data is always copied. There are no links or such happening

```
int a;  
int b;  
a = 20;  
b = a;  
a = 10;
```

CuSn

Exercise 2. Create a simple program that calculates the volume of shipping containers and determines how many boxes of sand can fit into the container if they are emptied in. A common standard size is 12m by 2.6m by 2.9m.

- a. Create variables for the length, width, and height of the shipping container. You should assume cm for the unit.
- b. Calculate the volume by multiplying these together and store it in a variable.
- c. Create variables for the length, width, and height of the shipping box. E.g., a U-Haul moving box is 45cm by 45cm by 41cm.
- d. Calculate the volume of sand in the box and store it in a variable
- e. Calculate how many whole boxes of sand can fit in the container and store the result in the container. Use the integer division.
- f. Since we are using integer division there is no fractional component. Calculate the remainder from the above division. What does this number tell you?
- g. Use the `println` command to output your results in any fashion that you like.

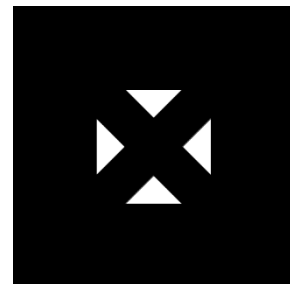
CuSn

Exercise 3. Using your knowledge of order of operations, by hand calculate the result from the following calculations. Do you get the same result as posted?

- a. $6+5*2+10 = 26$
- b. $10*10+10*10 = 200$
- c. $(1+1)*(2+2) = 8$
- d. $6\%2\%2\%2 = 0$
- e. $1\%2 = 1$
- f. $1+2-3*4/5\%6 = 1$

Ag

Exercise 4. This is a tedious exercise, but will really give you a lot of practice with the coordinate system and using variables. You will make a program to generate the crosshairs on the right, based on three initial variables: `spacing`, `crossX`, and `crossY`. The `x` and `y` is the center of the cross hairs, and the `spacing` defines how large it is.



- a. Create the top center triangle first. Create 6 variables, two per point. For example, `t1LeftX` and `t1LeftY` to represent triangle 1's left point X and Y position.
 - i. For the `x` coordinates, the center triangle point is at the `crossX`. The left and right are offset by `crossX` plus or minus the `spacing`.
 - ii. For the `y` coordinates, the center triangle point is offset from `crossY` by the `spacing`. The top two points are offset by $2*spacing$. Hint: the `y` coordinates of the top two points are

the same, so once one is calculated, copy that result into the other variable.

- b. Then, create the bottom triangle in a similar fashion. Notice that the x coordinates of the bottom triangle are the same as the top. Do not recalculate them, but instead copy them from the first triangle.
- c. Do similar operations for the left and right triangles.
- d. Try changing the size and position of your crosshair to make sure your variables and calculations are working properly!



Exercise 5. Copy-paste the cat example from the unit, where you can change variables to move it around. You will add one more functionality: scale. By changing a variable, try to make it so you can shrink or grow the head. There are a lot of ways to do this, my steps is just one way.

- a. Create a variable called `scale`, and set it to the size of the head (300 in this case). Calculate the other sizes (nose, pupil, etc.) based on this. E. g., the pupil is currently one twentieth of the head size and the nose is 1 tenth
- b. Here is the tricky part: all of your offsets, for the whiskers, eye locations, etc., will need to change. You already have the numbers on the 300 scale for all components, but will need to think about how to have them scale automatically.
 - i. Hint: your numbers (such as the whiskers being 25 or 50 offset) are on the 300 scale. How can you change scales? Careful, with integer division you cannot do straight percentages.



How did you do?

Learning Objectives

How did you do? Go back to the beginning of the unit and check how you measure up to the learning objectives.