Creative Code Lessons For Ikamva Youth

By Lyndon Daniels

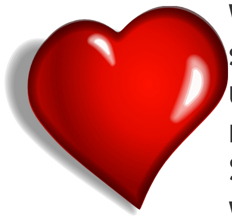
Based on Marion Walton's Training Resources at
https://ikamvacodes.wordpress.com/creative-code/
CREATIVE CODE is a Dr Marion Walton initiative

What is code?

Although you might not realize it, you already know the answer to this question. As you have been using and learning codes for most of your lifetime.

This course is all about understanding the different codes we use to communicate with computers, mobiles phones and each other. But before we get into creating our own code lets have a look at what codes are useful when talking to computers and how we can use what we already know to learn more about them.

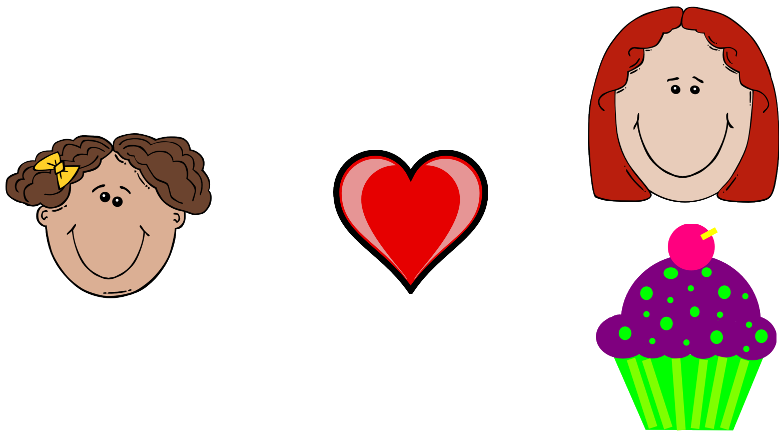
Symbols



We've all used symbols before to express ourselves. For example, have you ever sent some one you like a picture of a heart? This picture is a symbol, and we use it to express feelings such as love or liking someone or something very much.
Symbols have their own meanings and we use those meanings to communicate with our friends by agreeing on what that symbol means.

Code

As we already know, symbols can be a quick and fun way of communicating with our friends. But what about when we want to express ourselves in ways that are more detailed or complicated?

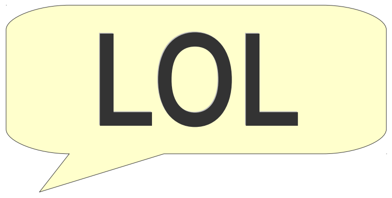


What do you think the picture above is trying to communicate with symbols?
Do you think it's saying *"I love my Mom as much as I love Cupcakes"*
or could it be saying *"I love my Mom and I like cupcakes alot!".*
Communicating with symbols alone can be quite confusing when we want to be more specific, but it certainly does make it easier to communicate more complex ideas when we combine symbols to form **codes**.

More Symbols with Many Meanings

We'll talk more about combining symbols to form codes a little later, but for now lets have a look at more symbols that you're already familiar with.

We all know that the symbol on the left is the South African flag, but have you considered that what it means for one person might be different to what it **means** to another?



Have you ever noticed that LOL is often used in text conversations, with different meanings to **different** people. Does it mean "laugh out loud" or does it mean "lots of love"?

Combining Symbols to Communicate Clearly

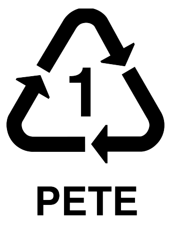
As we discovered previously, symbols by themselves can become a little confusing especially when we need to communicate clearly.
As a result, using symbols together can help with explaining the details of what we are trying to communicate but as we've already seen this is still not enough when we need to be perfectly clear, and when we communicate with computers it's very important that we are being **perfectly clear**.

If we really wanted to be perfectly clear and make sure we are being understood, we would need to provide a **context**.

Morse Code uses combinations of 5 different symbols to form it's code. A dot an a dash are easy to recognize as symbols in this illustration, but have you considered that the spaces between the dots and dashes can also be thought of as symbols.

The Context of Codes

A context is used to describe the main idea of what we are communicating. For example, when you see the following graphic on a plastic bottle you could describe it as having three different symbols.

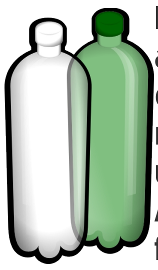


1. It has arrows

1 2. It has a number

PETE 3. It has a name

Finally, because this image appears on a plastic bottle it has a context. That context is "Recycling", and as a result we know that the meanings of the symbols used within this context are all about Recycling.

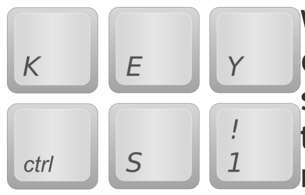


It is because of the context, that we know the number 1 is not referring to someone's age but it refers to a category that this type of plastic bottle falls into for the purposes of recycling. It is also because of the context that we know that the word PETE does not refer to someone's name but in fact it symbolizes what this plastic bottle is made up of (Polyethylene Terephthalate).
As you can see context is very important when communicating with code, it can go so far as to actually change the meanings of the symbols we use.

Communicating in Code with Computers

Symbols, **code** and **context** are equally as important when we need to communicate with computers.
Fortunately the symbols we use to communicate with a computer are often in the form of words you are already familiar with such as,

`width`	`setup`	`draw`	`color`	`triangle`
`height`	`curve`	`box`	`image`	`rotate`
`background`	`red`	`green`	`blue`	`alpha`



When we communicate with a computer these words symbolize commands that we use to tell a computer what we would like it to do. We string these commands together to form full lists of instructions and it is these instructions that we create to communicate with computers, that are referred to as **`code`**.

Exercise 01

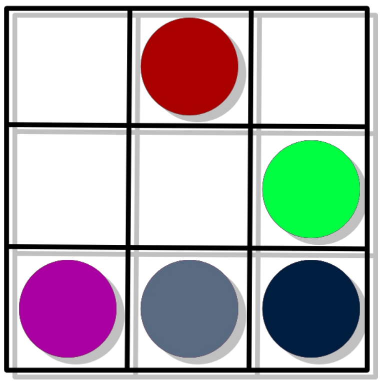
Create Your Own Pixel Art with the Secret Shape Game

In this exercise you will need

- A Pen and Paper
- Various Bottle Tops
- Two Teams

  Each Team has a person that will be the **Computer** and another person acting as the **Coder**.

The Game Starts with the Coders.



Your job as a Coder is to write a **list of instructions** on a piece of paper that will be handed over to the other team. No Drawings! These instructions must explain how to create a Secret Shape using the Bottle tops.

Making the Secret Shape

When the team member acting as the computer receives the instructions, she must make the Secret Shape by **following the instructions**. The Coders are not to help the Computers.

The Winner will be Judged by

Which team **finishes first**.

Which team has the **most beautiful shape**.

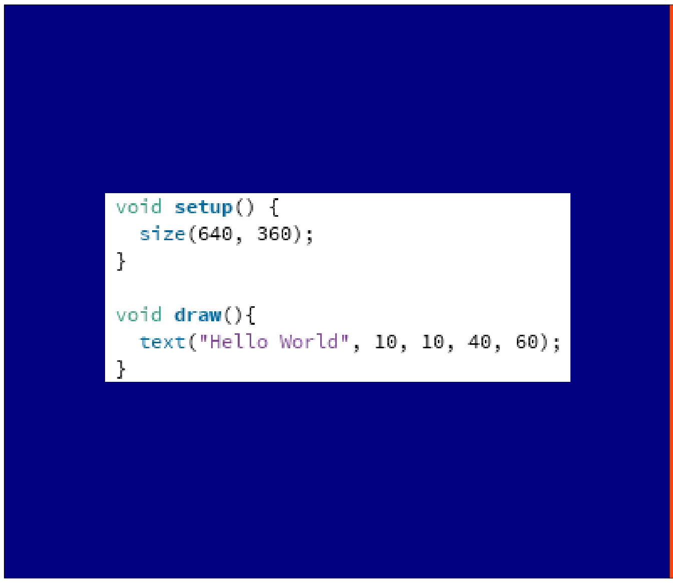
Your First Steps in Programming

When we talk to other people we need to communicate with them in a language that they can understand, as well as ourselves. Computers are no different in this way, because we communicate with computers in languages that are understood both by computers and ourselves.

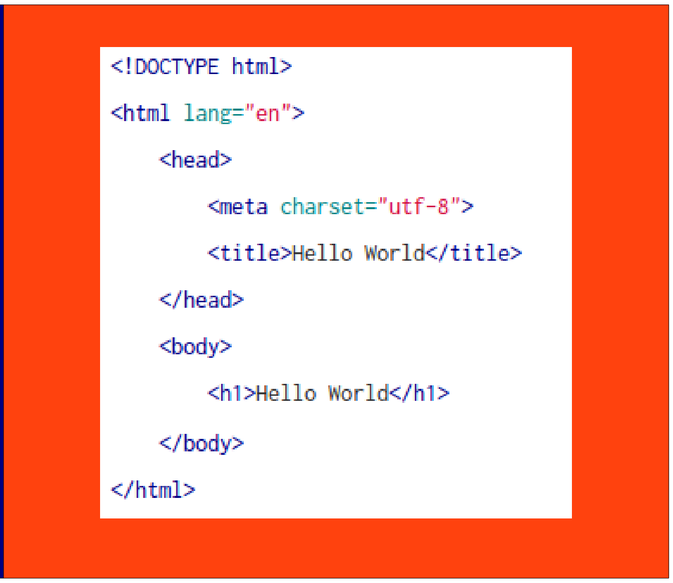
Computers also have many different languages that we can use to communicate with them such as HTML, JavaScript and Processing to name a few. This is very similar to the way in which many people we know also use different languages to communicate with each other such as Xhosa, English, Zulu and Afrikaans.

Processing is one of the many different languages we use to communicate with a computer, and is also the language we will be learning in this course to create code. You don't need to learn every different type of computer language, just the one's that will help you complete your tasks. These tasks will always have a context such as making a Web Site has the context of Web Development or making a Mobile App has the context of Mobile App Development. As the context changes for development so too will the **Computer Programming Languages** used.

Processing and HTML5 are both Computer Programming Languages, but because they are used in different contexts the languages look and act very differently.



```
void setup() {
  size(640, 360);
}

void draw(){
  text("Hello World", 10, 10, 40, 60);
}
```



```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>Hello World</title>
    </head>
    <body>
        <h1>Hello World</h1>
    </body>
</html>
```

The program on the left is coded in Processing and the program on the right is coded in HTML5. Although both programs do the same thing when they are run, which is, print the words "Hello World" to the screen, the way in which the same result is achieved differentiates substantially from one computer programming language to another because of the language's context.

The NameTag App

The task of creating code is known as **programming** and it involves creating a list of instructions that tell a computer what we would like it to do. These instructions are called a Computer Program or just simply a **Program** and will eventually become an application more often referred to as an **app**, just like the apps you have installed on your phone such as games, web browsers and text editors to name a few of the many different types of apps.

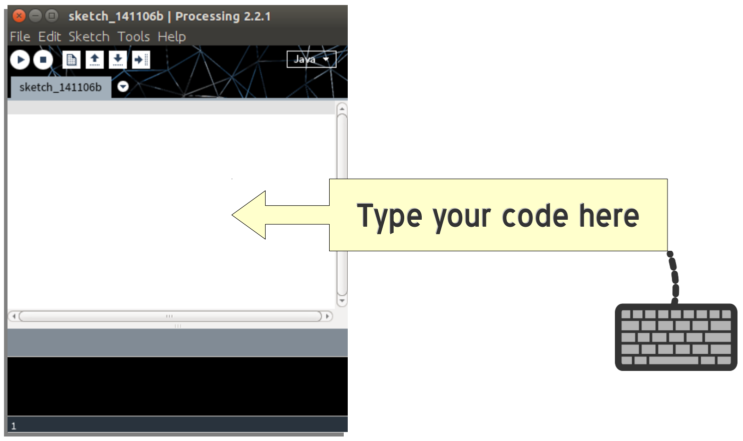
We've all used apps before either on our phones or on a computer, so lets have a look at how easy it is to create our own simple app in Processing that will allow us to make a nametag.

Although we will be making an app we will still need software and a computer or a phone to help us do this. Processing is a special type of application that helps us make our own apps.

Processing should already be installed on your computers, if not go to processing.org to download and install it. Ask your teacher for help if you're not sure how to do this.

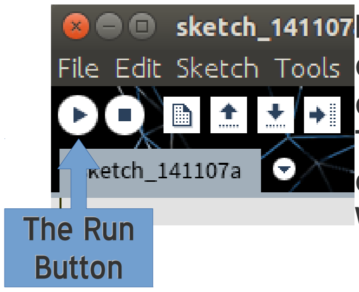
Lets Start Processing

Launch the Processing application and you'll notice that it immediately allows you to start typing in the white text area. This is the area that you will add your code to.


sketch_141106b | Processing 2.2.1
File Edit Sketch Tools Help
Java
sketch_141106b
Type your code here

Exercise 02

Create Your Own Name Tag Art with the NameTag App



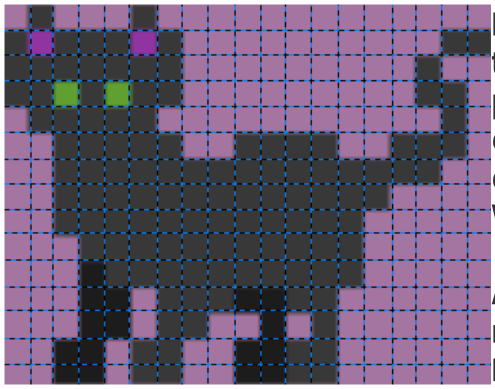
Below you will find the code for the NameTag app. All you need to do is copy and paste the code into Processing, click the run button and start creating your custom nametag.
Try pressing the b, r, w and s keys for variations. Then have look at the code and try changing the numbers where you see the fill() command. What happens when you run the program with the changes?

```
void setup() {

      size(480, 120);
      smooth();
      background(232,28,28);
}

void draw() {
      if (mousePressed) {
            fill(7,124,20);
      }else{
      fill(255);
      }

      if ((key == 'b') || (key == 'B')) {
            fill(0);
      }
      if ((key == 'r') || (key == 'R')) {
            fill(255,0,0);
      }
      if ((key == 'w') || (key == 'W')) {
            fill(255);
      }

      ellipse(mouseX, mouseY, 20, 20);
}

void keyPressed() {
      if ((key == 's') || (key == 'S')) {
            selectOutput("Select a file to write to:", "fileSelected");

      }
}

void fileSelected(File selection) {
      if (selection == null) {
            println("Window was closed or the user hit cancel.");
      }else{
            println("User selected " + selection.getAbsolutePath());
            save(selection.getAbsolutePath());
      }
}
```

Shapes and Coordinates

Pixels

Whether the screen you are using is on a mobile phone, a desktop computer, TV or any other digital device that screen is made up of pixels. Pixels are tiny dots that cover the screen, and store color information. As a result when many pixels are placed next to each other they can appear to form an image.



Every digital image is made up of pixels. They appear here as the blocks making up this image of a cat. This image is 19 pixels wide and 15 pixels high. As you can see it's not a very detailed picture of a cat, that is because there are not enough pixels to make up the missing details in the image. We call this a **Low Resolution Image**.

A more detailed picture of a cat would be made up of many more pixels, if the image was a digital photograph it would be a **high resolution image** consisting of thousands of pixels.

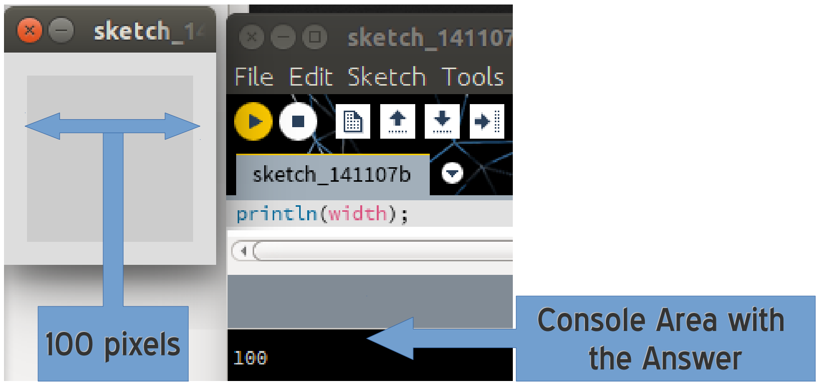
When we create code we need to be aware of how many pixels are available to us. Knowing this information will help us to draw shapes and other objects to a screen.

Processing makes it really easy for us to find out how many pixels we can use in our program by providing us with two special keywords width and height.

For example if we wanted to know how many pixels made up the width of our app we could ask Processing with a command such as the following,

```
println(width);
```

You can copy and paste this code into an empty processing document. Then hit the Run button to get the answer.
You'll notice that the answer appears in the bottom of the Processing window, we call the Console Area.



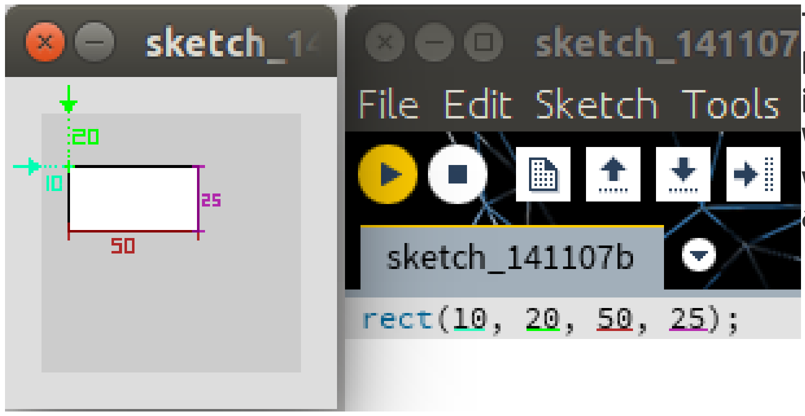
```
println(width);
```

```
100
```

When we know how many pixels are available to us, we can use this information to set coordinates for drawing shapes with code.

For example if we want to draw a rectangle in Processing we would need to first tell Processing where to place the rectangle, then how wide and tall the rectangle is. For example the following code will draw a rectangle,

```
rect(10, 20, 50, 25);
```



```
rect(10, 20, 50, 25);
```

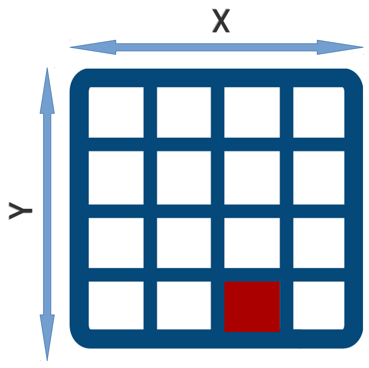
To draw a rectangle we use the keyword rect(). This type of command is known as a **function**.
We can control the rectangle function with four numbers that we call arguments.

- The first argument "10" is how many pixels the rectangle is from the left edge of the app screen.
- The second argument "20" is how far the rectangle is from the top of the app screen.
- The third argument "50" is how wide the rectangle is.
- The fourth argument "25" is how high the rectangle is.

As you can see arguments are really important for modifying a function's behavior.

Coordinates

As we already know the pixels that make up our screen are placed next to each other, the placement of these pixels therefore forms a grid.



In this image of a grid we have a screen that is made up of 16 pixels. We call this a 4 x 4 (four by four) grid, because it is 4 pixels wide and 4 pixels high.

In programming we call the width of a grid the X axis and we call the height of a grid the Y axis, therefore the coordinates of the red pixel is (3, 4) that is 3 X and 4 Y.

Variables

A variable is a container for values that we can change or store in our programs.

Let's think about the different ways in which you can adapt or customise a simple Pacman character. You might want to:

- Draw Pacman in different places in the sketch window.
- Change the color of Pacman.
- Make Pacman face left or right
- Turn Pacman into Ms Pacman by adding a little bow.
- Make Pacman bigger or smaller by increasing or decreasing the radius used to draw the arc (part of a circle).

We will use variables to create all these different versions of Pacman.

Integers store numbers

An integer is a whole number or zero. An integer can be negative, positive or zero. If we want to change the position where we draw Pacman we need integer variables. We need two integers because we want to be able to place Pacman on both the x and y axes.

```
int pacX = 50;
int pacY = height/2;
```

Here we are using two integer variables to store the position of our pacman character. In the first case we specify a particular value (pacX), and in the second case (pacY) we calculate a value – height/2. There 'height' is a also a variable – a special one used to store the height of the sketch. Notice that in each case the variable consists of three components:

- The type of variable (eg int).
- The name of the variable – imagine this as a kind of box or container (eg pacX, pacY).
- The value of the variable – imagine this as something which is stored in the box (eg 50, height/2).

We also use an integer to store the radius of Pacman. This radius controls the size of the circle (arc) that we draw.

```
int radius = 30;
```

Booleans store true or false

Sometimes we need a special kind of variable, one which can store only two values – true or false. This kind of variable is known as a Boolean. Here we use it to decide whether Pacman will be male or female.

```
//boolean to set the sex of the pacman characters
Boolean female = true;
```

Color variables

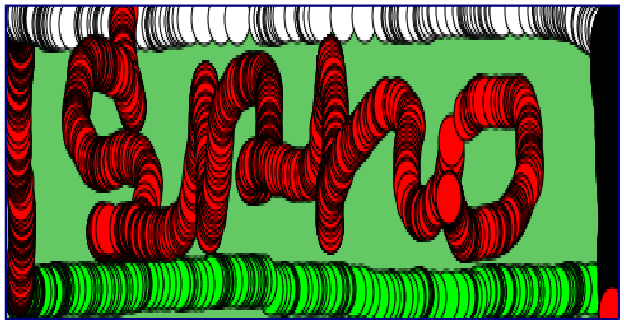
We can also use a special kind of variable (known as an object) to change colours. These are called 'color' – notice the U.S. spelling! Colors are a bit different to simple integers, because we use one color to store three or four integers. These work together to create a colour. As we have discussed in a previous lesson, each colour is created by combining values for

- red
- green
- blue
- (optionally) alpha or transparency

```
color pink = color(255, 134, 241, 125);
```

Conditionals

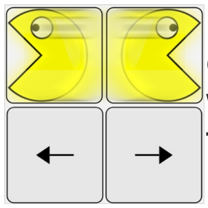
So far, the Processing commands that we have written have been executed one after the other, in the order they are written. This is fine for very simple programs, but mostly we want certain commands to be executed and others not to be.
To skip some commands under certain conditions, we need to have special commands called control statements which control the order in which our commands are executed in a program. The most widely used control statement is called an 'if statement'.



In this example by Sipho, he used an 'if' statement to determine what colour is drawn when the user moves the mouse around.

Sipho Msai Ngqayimbana

Checking the value of variables



If statements are used to check the value of a variable and then execute a different set of commands depending on the value of the variable. For example, if we set a Boolean variable "direction" whenever someone presses an arrow key, then we can check that variable before we draw the Pacman.

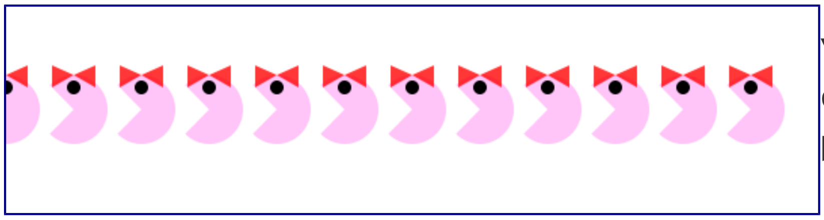
```
/*this checks the value of the variable 'direction',
and whether it's true that it equals zero*/

if (direction==0)
      {
        //if it is true, then draw pacman arc facing right
        arc(pacX, pacY, radius,radius, radians(45), radians(315));
      }
else //if the condition is not true,
      {
        //if it is not true, then draw pacman arc facing left
        arc(pacX, pacY, radius,radius, radians(225), radians(360+135));
      }
```

You can play around with the code for this sketch in this sketchpad.

```
int pacX=50;
int pacY=height/2;
color pink = color(255,134,241,125);
int radius = 30;        //set the radius variable for the pacman characters
Boolean female = true;  //boolean to set the sex of the pacman characters
int direction=0;
PFont font;

background(0);
size(100,100);
smooth();
noStroke();
fill(pink);
  if (direction==0)
  {
    //draw pacman facing right
    arc(pacX, pacY, radius,radius, radians(45), radians(315));
  }
else
  {
    //draw pacman facing left
    arc(pacX, pacY, radius,radius, radians(225), radians(360+135));
   }
//draw eyes
fill(0);
ellipse(pacX,pacY-radius/3,radius/5,radius/5);
if (female == true)
  {
    //draw ribbon for ms pacman
    fill(255, 0, 0, 200);
    triangle(pacX,pacY-radius/2,pacX+radius/3,pacY-(radius/3)*2,pacX+radius/3,pacY-
radius/3);
    triangle(pacX-radius/3,pacY-radius/3*2,pacX,pacY-radius/2,pacX-radius/3,pacY-
radius/3);
  }

fill(0, 102, 153);

//move pacman around the screen
  if (keyPressed && (key == CODED)) {     // If it's a coded key
    if (keyCode == LEFT) {                // If it's the left arrow
      pacX-=5;
      direction =0;
    }
    else if (keyCode == RIGHT) {          // If it's the right arrow
      pacX+=5;
      direction=1;
    }
    if (keyCode == UP) {                  // If it's the left arrow
      pacY-=5;
    }
    else if (keyCode == DOWN) {           // If it's the right arrow
      pacY+=5;
    }
  }// end if keypressed
```

Using a "for" Loop to Create Patterns

In this example, we will learn how to draw a pattern by repeatedly calling a function. In a previous example, we saw how we could draw lots of different versions of pacman by calling a function several times and changing the parameters. We draw the same shape over and over again without having to cut and paste lots of function calls. We control how often it repeats, and how much the position of the shape changes with a For Loop.



You can change the pattern by changing the size and colour of the pacman.

If you want to see how the Ikamvacoders modified the “for” loop, take a look here.

Basically the for loop lets you control a repeated action, such as drawing a pattern. We're using the drawPacman function we created in a previous lesson.

```
for(x=0; x<width; x+=radius){
  pacman(x, y, pink, false, radius, 1);
  }
}
```

There are three parts you need to understand to create a For Loop:

1. init – this is the starting point of your value. In our example, the init value of x is 0 so that we can start drawing our pacman pattern on the left edge of the sketch. Can you work out how you could get the first pacman to display properly?

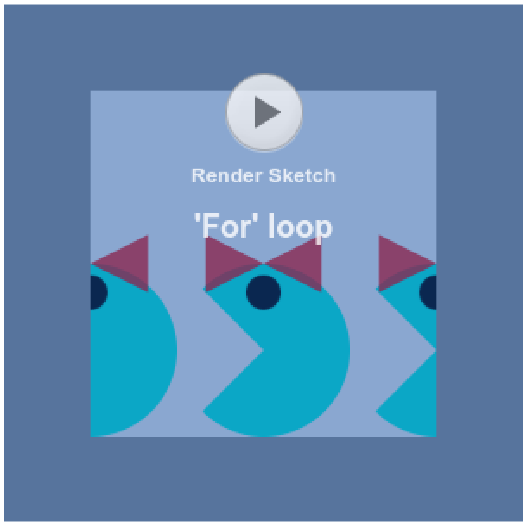
2. test – This is a statement about your value that the program must test and decide whether it evaluates true or false. In our example, the test works out whether x is still smaller than the width of the sketch (because that is where we want to stop drawing the pattern). This test must be true for the For loop to continue.

3. update: the update changes the value you're working with, In our example we add the radius of pacman to the value of x every time. x+=radius means the same as x=x+radius. This way our pacmen don't overlap.

Each part must be separated by a semicolon (;). If the test evaluates true, then the For loop executes the statements in the curly brackets below. If it evaluates false, then it doesn't execute them and the loop stops. You need to ensure that you don't create an endless loop by setting a test that will never become false. Here's a more precise description from the Processing website, where you can see other examples of For Loops:

1. The init statement is run.
2. The test is evaluated to be true or false.
3. If the test is *true*, jump to step 4. If the test is *false*, jump to step 6.
4. Run the statements within the block.
5. Run the update statement and jump to step 2.
6. Exit the loop.

You can see the code below, or try out the code in a sketchpad. Can you change the pattern? See whether you can figure out how to get a vertical pacman pattern instead of a horizontal one.



```
//Draw a row of pacman characters using a function

float x;
float y;
color pink = color(255,134,241,125);
int radius = 50;        //set the radius variable for the pacman characters
Boolean female = true;  //boolean to set the sex of the pacman characters

void setup(){
      size(600,radius*3);
      x=0;        //set initial values for the pacman coordinates
      y=height/2; //position pacman correctly for background graphic
      }

//the draw() function executes repeatedly to create the animation
void draw(){
      smooth();
      background(255);  //draw background
      //use for loop to draw series of pink pacmans across the screen
      for(x=0; x<width; x+=radius){
            drawPacman(x, y, pink, false, radius, 1);
            }
      }

void drawPacman(float pacX, float pacY, color clr, Boolean ms, int pacRadius, int
direction){
      fill(clr);
      noStroke();

      if (direction==0){
            //draw pacman facing right
            arc(pacX, pacY, radius,radius, radians(45), radians(315));
      }else{
            //draw pacman facing left
            arc(pacX, pacY, radius,radius, radians(225), radians(360+135));
      }
      //draw eyes
      fill(0);
      ellipse(pacX,pacY-radius/3,radius/5,radius/5);
      if (female == true){
            //draw ribbon for ms pacman
            fill(255, 0, 0, 200);
            triangle(pacX,pacY-radius/2,pacX+radius/3,pacY-
(radius/3)*2,pacX+radius/3,pacY-radius/3);
            triangle(pacX-radius/3,pacY-(radius/3)*2,pacX,pacY-radius/2,pacX-
radius/3,pacY-radius/3);
      }
      //save a screenshot of the image
      if (mousePressed) {
            if (mouseButton == RIGHT) {
                  save("pacman_pattern.png");
            }
      }
}
```

Exercise 03

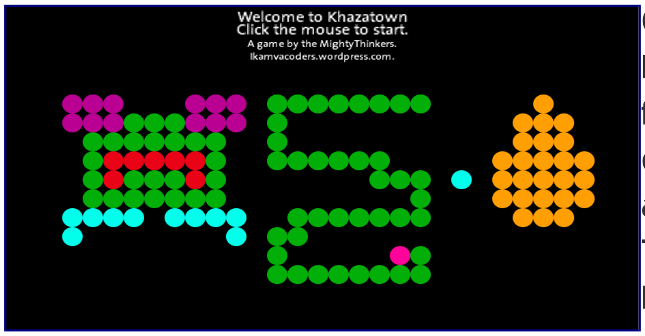
Make your own version of a Pacman Mod

In this exercise we'll be using the Pixel Art Creator to design characters. Your challenge is to think of game stories which could fit the Pacman structure. Here there are three basic roles:

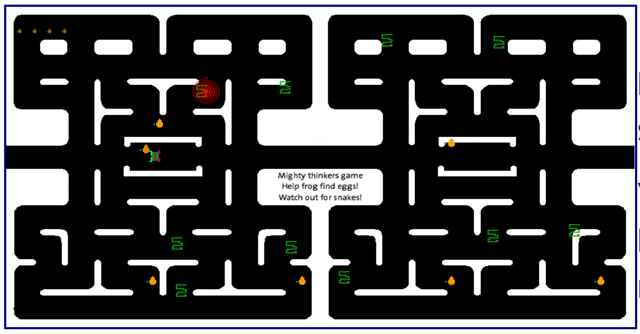
1. The hero or she-ro (Pacman role)
2. The opponent (Ghost)
3. The target or goal (Pacman's pills)

Here's an example of some previous Mods.

The Mighty Thinkers - Welcome to Khazatown

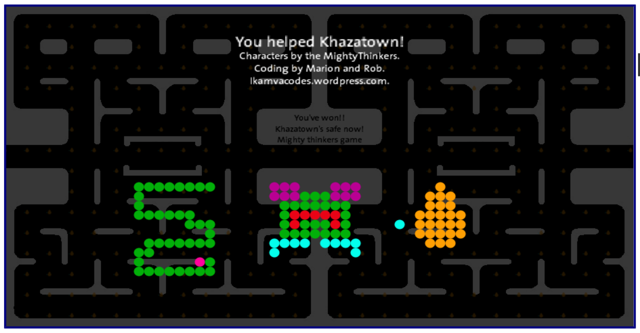

Welcome to Khazatown
Click the mouse to start.
A game by the MightyThinkers.
Ikamvacoders.wordpress.com.

One of our groups designed three new characters. In the Mighty Thinkers' Pacman mod, you are a frog (hero), and you have the goal of collecting golden eggs (target) while guarding yourself against the snakes (opponents) in the vicinity. These are the three characters in Welcome to Khazatown – frog, snake, and egg


Mighty thinkers game
Help frog find eggs!
Watch out for snakes!

Playing Welcome to Khazatown: Watch out for the snakes!

You can download the game and try it yourself. Let us know what you think and show us any mods (modifications) that you make!

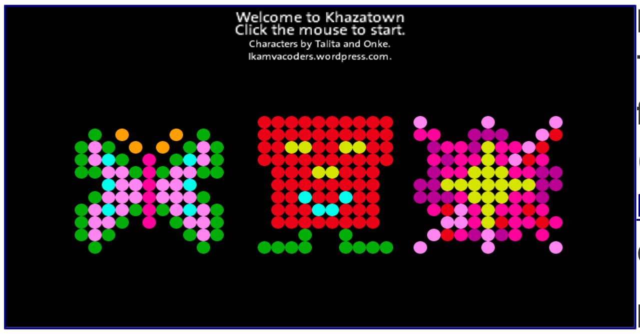

You helped Khazatown!
Characters by the MightyThinkers.
Coding by Marion and Rob.
Ikamvacodes.wordpress.com.

You've won!!
Khazatown's safe now!
Mighty thinkers game

Download *Welcome to Khazatown* as:

- Processing source code
- Windows application
- Linux application
- Mac OS application

If you're not ready to change the game code, why don't you try using your own pixelart characters in our game.


Welcome to Khazatown
Click the mouse to start.
Characters by Talita and Onke.
Ikamvacoders.wordpress.com.

Here are characters designed by Talita and Onke – Tally the butterfly (hero) transfers pollen from one flower (goal) to the next, while the evil dustbin (opponent) gets in her way. Onke's dustbin started life looking more friendly, but now he is Tally's opponent.

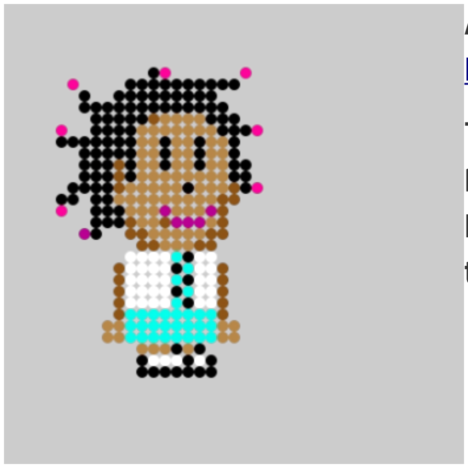
Pixel art characters designed by Talita and Onke

Exercise 04

Design Your Own Mario Mod Characters

In this exercise we used a Processing.js library built by Pomax. You can try Pomax's tutorial if you want to make your own Mario game.

KHAZATOWN BLUES



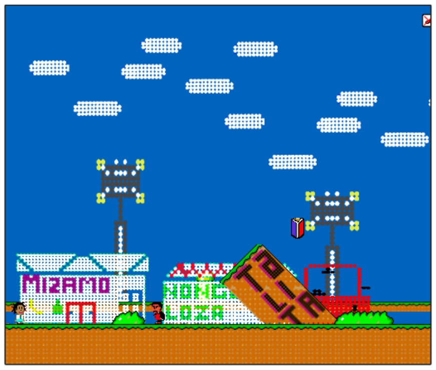
A game by Lisa, Talita, Ludwe, Lwazi, Onke and Vuyani. Ikamvacoders from Ikamva Youth.

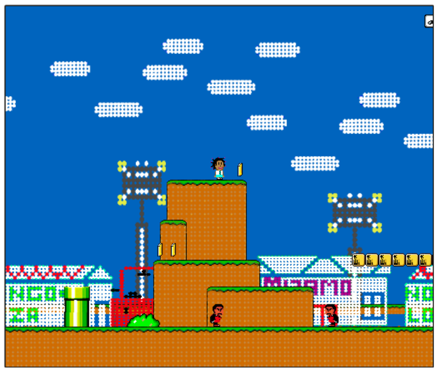
The name of the game is Khazatown Blues. Going to school in khazatown is not a child"s play. You have to watch your back!! Bullies are out to GET YOU therefore make sure you GET all the life supporters to survive in the game.

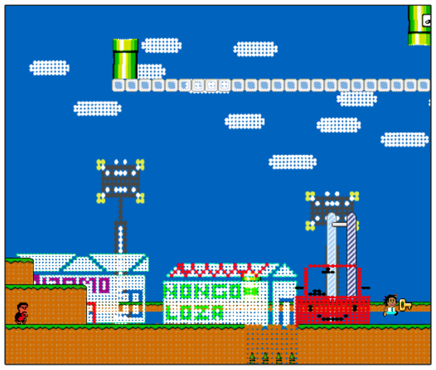
- EAT AS MANY BOOKS AS YOU CAN!!
- In this game its all about avoiding your enemies the bullies and the evil principal.

isiXhosa

Funeka ubaleke abaxhaphazi abanomona ngemfundo yakho Kunye nenqununu engcolileyo engxothayo xa ufike emva kwexesha. Ukuze uhlale uphilile funeke utye incwdi ezininzi kangangako ukwaziyo. Funeke ungavumeli ungene kwindawo eneziyobisi ngoba uzofa.







Play the game Now!