CODE BANK

- size() --> adjusts size of canvas
- ellipse(i1,i2,i3,i4) --> draws ellipse middle at (i1,i2) with width i3 and height i4
- background() --> background colour of canvas
- noFill() --> shapes are see through
- fill() --> fills shape with colour
- point(i1,i2) → colours pixel black
- stroke() → outline of shape colour
- noStroke() → no outline
- ellipseMode() → changes the strategy which Processing uses to draw the ellipse
- arc() → draws an arc
- frameCount

Processing Reference: https://processing.org/reference/

Foundation of Programming – 28/2/20

Values and Types

- Values are grouped into types
  - int: whole numbers
  - float: numbers that may have decimal parts
  - char: single characters that might appear in text
- A *int* can easily be converted to a *float* but a *float* does not always have a corresponding *int*.
- If you see '1' processing considers it an *int* but if you see '1.0' processing considers it a *float.*

Algorithms

- An algorithm can be considered as a task.
- We define every task according to its:
  - The **Purpose** of a task is its *Name*
  - If a task has **Inputs** it is *Informed*
  - If a task has **Globals** it is *Intertwined*
  - If a task has **Effects** it is a *Changer*
  - If a task has **Outputs** it is a *Producer*

Algorithms in Processing

- We never write code from scratch, but rather use existing functions such as *line().*

EXAMPLE with a Basic Function

Draw Rectangle

- **Inputs**: *float, float, float, float*
  - First two give coordinates of top left corner
- **Global**: (none)
- **Effects**: draws a rectangle as specified
- **Output**: (none)

Mow the Lawn

- **Purpose**: make the grass short
- **Input**: area to mow lawn
- **Global**: grass length
- **Effects**: the grass in that area is shorter, the air smells of two-stroke
- **Outputs**: none

Creating your own function that draws a green rectangle:

```
void drawBox() {
  fill(0,200,0);
  rect(40,30,10,15);
}
 void setup() {
   drawBox();
 }
```

We can then add input parameters (x and y) to make the rectangle have a variable position.

```
 void drawBox2(int x, int y) {
   fill(0,200,0);
   rect(x,y,10,15);
 }

  void setup() {
   drawBox2(10,20);
   drawBox2(70,40);
 }
```

- The function *setup* is needed because it is the function that Processing runs.

Lecture 3 – 3/3/20

- We can colour each pixel individually using
- A program is a sequence of commands, and a programming language is an automated way of giving commands to your computer.
- ellipseMode() changes the strategy which Processing uses to draw the ellipse
- When drawing arcs, put ellipseMode into CENTER
- A **statement** is an instruction for a computer to do something
- An **expression** is simply just a value

Variables and Conditionals – 8/3/20

Variables

- Variables are active values that are stored within the RAM of a system.
- They can either be **built in** (e.g. mouseX) or **user-defined**.
- User-defined variables occur in 3 steps
    - Declare the variable (top)
    - Initialise the variable (setup)
    - Use the variable (draw)

- When declaring the variable, it must be in the form *type name;* and it is important to choose the correct data type (e.g. int, float).
- The name must not coincide with any in built functions as this can cause errors or confusion.
- Initialising the variable requires a line of code called the *assignment operation*, which assigns a certain value to something else.

Incrementing a Variable

- The program runs *draw* over and over again, at a specified frame rate.
- In order to increment a variable, add one to the integer variable within draw, as this will constantly run.

```
void draw() {
  background(50);
  fill(255);
  ellipse(circleX,180,24,24);

  circleX = circleX + 1;

}
```

- In order to double the speed, add 2, triple speed, add 3, so on…
- In order to half the speed, we must add 0.5. However, an integer value cannot be a decimal value, and thus we must define the variable as a *float.*
- NOTE: Define all variables as *float* in general!

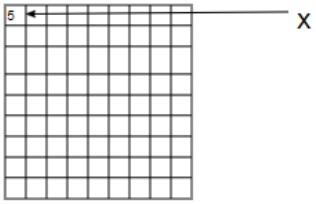
Using random()

- random() is a function call and can be used within an assignment operation.
- Random(100), will return a random number between 0 and 100 (not including 100). Random (50,100) returns a random number between 50 and 100.
    - i.e. random(min,max);

How Memory Works

- When processing is running, it has access to a bank of memory and it might put things in there or read things from there.
- We visualise the memory bank as a grid of buckets (or slot in memory). Each bucket may hold a value and a program with no variables will look like a grid of empty holes.
- To fill up buckets, we must put values into them. This is done by naming the slot and then filling it, with the use of **variables**.

```
1  int x;
2  x = 5;
```



- We don't know exactly what memory slot will be used, but we do know it will be called 'x'.

Debugging in Processing – 10/3/20

- Click the debug icon the begin the debugger.
- A debugger will set *break points* at certain lines in your code so you can pause at these lines and investigate.
- At the break point, you will be able to see the values of all the variables.
- Using the *step* button will move through the code line by line, and the *continue* button will go to the next break point.
- We can also use the println() function to print certain values in the console. This allows us to see the actual values and determine any errors in the code.

Conditions – 10/3/20

Boolean Expressions

- Conditional statement: using if()
- The expression that goes into the brackets of an *if* statement is called a Boolean expression. We create these expressions using relational operators:
    - Greater than: >
    - Less than: <
    - Greater than or equal: >=
    - Less than or equal: <=
    - Equal: ==
    - Not Equal: !=
- We generally use variables within the if statement (rather than hard coded numbers).

```
if(y>(height+25)) {
  y=0;
}
```

If, Else if, Else

- Within an if statement, if the condition is met, that code is executed. If the condition isn't met, and the next condition within the *else if* statement IS met, that code will execute, and so on in order from top to bottom.
- An *else* statement will only run if none of the previous conditions are met. i.e. it will run after all conditions are tested and all of them are false.

```
if (mouseX > 500) {
  background(255,0,0);
} else if (mouseX > 400) {
  background(255,255,0);
} else if (mouseX > 300) {
  background(255,0,255);
} else if (mouseX > 200) {
  background(0,0,255);
} else {
  background(0,255,0);
}
```

Logical Operators

- AND and OR are logical operators.
- These are used to join multiple Boolean expression.

- AND is written as && within processing.
- AND statements will only evaluate to true if both statements are true.
- OR is written as || within processing
- OR statements will evaluate if either statement is true.
- There is also the NOT operator which is written as !. It is essentially the inverse of the conditional statement.

Boolean Variables

- A *Boolean* is a variable type (just like *int* or *float*). You can set the value of a boolean to either true or false.
- Boolean variables can be placed inside conditional (if) statements as they evaluate to either true or false.

Loops – 3/4/20

- Loops are similar to conditions except that after every iteration of the loop, the expression is checked again.
- Loops are statements, not expressions. This means they do not have equivalent values.
- Both loops and if statements are called **control-flow** statements because they change the flow of the program from top-to-bottom, to something more complex.

While Loop

- Example:
    - ```
      x = 0;
      while (x < width) {
              //execute this code
              x = x + 20;
      }
      ```
- **Note:** we require 3 basic elements
    - Initialisation condition (*x = 0*)
    - Boolean expression (*x < width*)
    - Incrementation operations (*x = x + 20*)
- An *if statement* can run either 0 or 1 times, whereas a *loop* can run as many times as you specify.
- Loops must always have an exit condition to ensure that they do not get stuck in an infinite sequence.

For Loop

- In a *for loop*, we have the three basic elements all in one line of code.
- Example:
    - ```
      for (int x = 0; x < width; x = x + 20) {
              //execute this code
      }
      ```
- This is still exactly the same as a while loop (just in a different format)

Logic Tables

- Logic tables are an effective way of tracing loops
- Guidelines for constructing a logic table:
    - Identify all variables involved in the boolean in the loop header.
    - Create columns for each of the variables identified
    - Create a column for the loop expression
    - Create columns for each variable modified in the loop (in the order they are modified)
- Example:
    - *int a = 6;*
      int result = 1;
      while (a > 0) {
      *result = result * a;*
      a = a – 1;
      }

a	a > 0	Result
6	True	1*6 = 6
5	True	6*5 = 30
4	True	30*4 = 120
3	True	120*3 = 360
2	True	360*2 = 720
1	True	720*1 = 720
0	False	Run rest of code

Functions – 24/4/20

- Examples of functions are *background(), rect(),* etc. These are all pre-defined by Processing.
- We can define our own functions that aren't already predefined by Processing.
- A function definition has three parts: return type, function name and arguments/parameters.
- The Syntax for a function is:
    - *returnType name (<parameters>) {*
      //execute this code when function is called
      }
- This syntax defines a function and it must be its own block of code.
- We can call our functions within draw.
- The syntax of a function call is *functionName(<parameters>);*
- Formal vs Actual Parameters:
    - Parameters appearing in the top line of a function definition are called **formal parameters** and are defined in the same way as any other variable declaration.
    - Parameters appearing a function call are called **actual parameters** and work like any other value in Processing.
- Two big advantages of functions are modularity and reusability.
- If a function doesn't return a value back, its return type is *void*.

Reusability with Functions

- We can declare variables as the parameters of our function (local variables)
- These are the arguments of our functions which we can change as we call the function within draw.
- This allows for the reusability of functions, without having to re write the same block of code multiple times.