IEEE
Computer Chapter
Southern Alberta Section

SCHULICH
School of Engineering
UNIVERSITY OF CALGARY

IEEE Women in Engineering
WIE
Southern Alberta Section

SCHULICH IGNITE 2019

Session Overview

- **Functions**

Functions

Pre-Defined Functions

So far, we've been using functions like *commands*:

```
println("Hello World");       // print something
ellipse(200, 300, 50, 50);    // draw an ellipse
fill(200, 120, 50);           // change the fill colour
```

They just do an action for us

Return Values

Some other functions *give us back* a **return value**

```
max(11, 13);      // returns the larger of the two numbers
str(893);         // converts to a String and returns it
random(10.0);     // returns a random float between 0 and 10
sort(array);      // returns a sorted copy of the array
```

Return Values

These can be used in place of any value just like a variable

```
int    number = max(11, 13);
String digits = str(893);
float  rand   = random(10.0);
int[]  array  = sort(array);


println( max(11, 13) + " is the bigger number");

fill( random(255), random(255), random(255) );
```

Functions

- Functions package code together into a reusable chunk with a name
- They let you split up your program into many smaller parts
- They make life easier by letting you reuse code, so that you don’t need to write the same things over and over again

Examples of functions

Take a vending machine for example.

It has a few different simple functions that need to be repeated over and over:

- Take in money and count it
- Take in the person’s choice of food
- Dispense the chosen item if there is enough money

Making our own functions

We can create our own functions too!
Here is the anatomy of a function:

```
type name(parameters) {

    // function body
    return value;
}
```

Function Names

All functions have a **name**.
This is what we use to *call* our function

```
type name (parameters) {

    // function body
    return value;
}
```

Return types

All functions have a **return type**.
This is what kind of value the function passes back

```
type name(parameters) {

    // function body
    return value;
}
```

- Return types can be: `int`, `float`, `String`, `Ball`, etc.

Returning Nothing

What if we want to return *nothing*?

Then our return type is...

void

Function Body

All functions have a **body.**
This is where we say what the function actually *does*

```
type name(parameters) {

    // function body
    return value;
}
```

Simple Example

- This is the simplest type of function
- It just *does* something
- It doesn't return anything (only **THE VOID**)
- It doesn't take in any parameters

```
void sayHello() {
    println("Hello, World!");
}
```

Return statement

If a function is not void, we need a **return statement.**
This is the part that actually returns the value

```
type name(parameters) {

    // function body
    return value;
}
```

Return Example

- These are functions with a return value
- It just returns an int
- It doesn't take in any parameters

```
int getMyAge() {
    return 20;
}
```

```
int twoPlusTwo() {
    int answer = 2+2;
    return answer;
}
```

Function Parameters

Functions can also take in **parameters**.
These are the values supplied to a function that get captured in variables.

```
type name (parameters) {

    // function body
    return value;
}
```

Parameters Example

- This is a function that adds two numbers together
- It takes in two int parameters
- It returns the sum as an int

```
int add(int a, int b) {
    return a + b;
}
```

Another Example

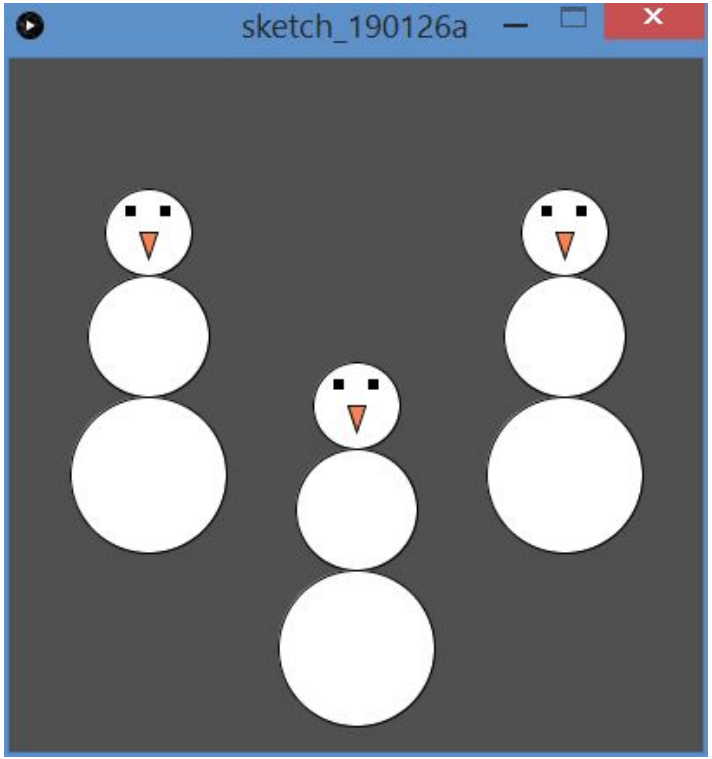
```
float calculateCubeVolume(float sideLength) {
    return sideLength * sideLength * sideLength;
}

float volume = calculateCubeVolume(2);


println("A cube with side length 2 has volume " + volume);
```

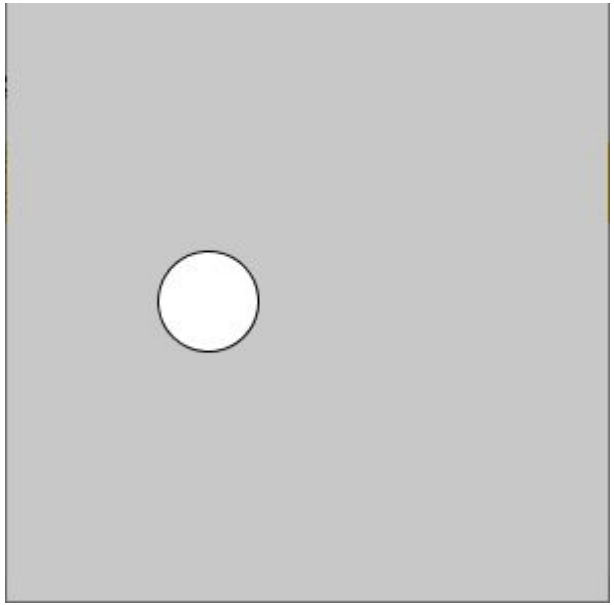
Exercise 1: Drawing a Snowman

- **Create** a function called drawSnowman() that draws a snowman on the screen
- The function should take the following parameters as inputs:
  - x-coordinate of the middle of the Snowman's head
  - y-coordinate of the middle of the Snowman's head
- **Call** your function in setup multiple times



Exercise 1: Making a Snowball Factory

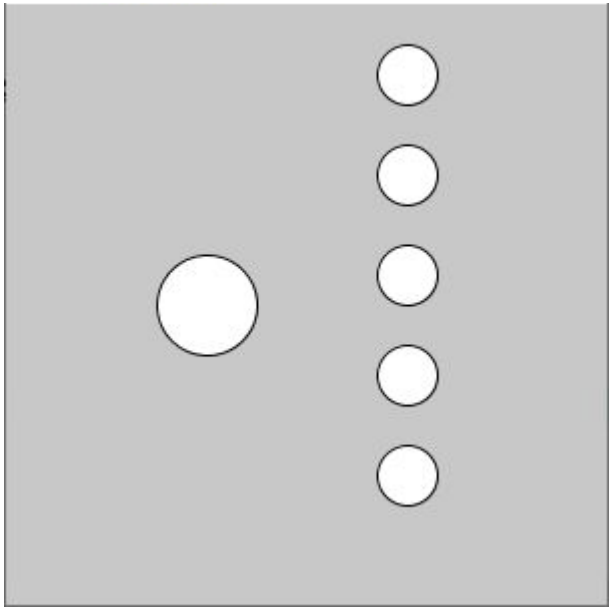
- **Create** a function called **drawSnowball** that draws a circle on screen
- The function should take the following parameters as inputs:
  - x-coordinate
  - y-coordinate
  - Size
- **Call** the function inside draw()



Exercise 2: Using the Snowball Factory

- Inside draw(), **call** your function **drawSnowball** again, this time using a loop

- **Bonus:** Change the function **drawSnowball** to make *all* the snowballs move to the right



Events

A **pre-defined** function like ellipse() has already been *defined* for us.
It is our job to *call* it if we want to use it.

An **event callback** is a function that is *called* for us.
It is our job to *define* it ourselves.

setup() and **draw**() are examples of event callbacks.
Processing calls **setup** once and then calls **draw** in a loop

User Events

There are other event callbacks that Processing will call (if we define them) when an **event** happens:

```
void mouseClicked() {
    // This happens when the mouse is clicked
}

void keyPressed() {
    // This happens when a key is pressed
}
```

mouseClicked()

Use **mouseClicked()** to run code every time the user clicks their mouse.

```
void mouseClicked() {
    println("Clicky!");

    // Example: Make the ball jump up when you click
    ball.speedY -= 5;
}
```

keyPressed() (and keyPressed)

Note:
key → letters
keyCode → arrow keys

Use **keyPressed()** to run code when a key is pressed

```
void keyPressed() {

  if (key == 'a') {
    println("a was pressed");
  }

  if (keyCode == LEFT) {
    println("left was pressed");
  }
}
```

Use the variable **keyPressed** to run code while a key is held

```
void draw() {
  if (keyPressed) {
    if (key == 'b') {
      println("b is held");
    }

    if (keyCode == DOWN) {
      println("down is held");
    }
  }
}
```

Exercise 3: Tying it all Together

- Create a function called **drawDartboard** that draws a dartboard on the screen
- Design the function **drawDartboard** so that it takes the following parameters as inputs:
  - Location of the dartboard
  - Diameter of the dartboard
- Call your function **drawDartboard** in the **draw** function multiple times, with different parameters that are stored in an array

Hint: Use a loop to draw all the circles of different sizes

Function members

- When you create a function inside a class, the function has a special name; it is now called a **member function**.
- This is great to use if you have a class that can do more than one thing.
- If you have, or want to create, a function that is related to a class, then it is best to make that function a function member for that class.