

COMP 1010- Summer 2015 (A01)

Jim (James) Young

young@cs.umanitoba.ca

jimyoung.ca

Hello!

James (Jim) Young

young@cs.umanitoba.ca

jimyoung.ca

office hours T / Th: 17:00 – 18:00

EITC-E2-582

(or by appointment, arrange by email)

User-defined functions-
make your own commands

user-defined functions

user-defined functions: code which the user can write once and then use over and over. E.g., random was written once and you can use it over and over by calling random(number)

create once and use many times!

Example: draw random squares

If the mouse is pressed, clear to white first

If a key is pressed, clear to black first

Random place, random size, random color

Global: max size

Tedius – repetitive

Let's make a new command called
`drawRandomSquare()`;

user-defined functions

require:

function name: the keyword used to invoke (use) the function. e.g., **ellipse** is a function name

some code: the processing code to run every time the function is invoked (called).

user-defined functions: syntax

```
void functionName ()  
{  
    ...//code;  
}
```

user-defined functions: syntax


```
void functionName ()  
{  
    ...//java code;  
}
```

```
void sayHello()  
{  
    text("Hello",100,100);  
}
```


user-defined functions: syntax

```
void sayHello()  
{  
    text("Hello",100,100);  
}
```

this is a the name of
a new command we
just created!



... in your other code
sayHello();

create outside the draw and start blocks!

note: user-defined functions created outside the other blocks

you can place them before or after them, it is a matter of style

jump around...

when a function is called, Processing remembers where it left off, and jumps to the function. When the function is done, it jumps back.

program flow

```
void sayHello()
```

```
{
```

```
text("Hello",100,100);
```

run each command in the function (compartment) in order

```
}
```

to run this command, jump up here! To the compartment!

program starts here as usual

```
void draw()
```

```
{
```

```
sayHello();
```

run this command first, as usual

```
sayHello();
```

when the function is done, come back here and continue where we left off

```
}
```

and run the function again!

Example: draw random squares

Let's make a new command called
`drawRandomSquare()`;

functions and scope

note: scope is the range within which a variable exists. Outside of that scope, you cannot access or work with that variable.

blocks define scope!

Functions each have their own scope – data in one method is not visible to other methods – try it



variable scope is limited to its function



```
void sayHello() {  
    text(100,100,message);  
}
```

```
void draw() {  
    String message = "Hey there!";  
    sayHello();  
}
```

note: a variable declared in one function is not **accessible** from another function!

you cannot read it, or change it.

No nesting of scopes here

the same variable name?

```
void sayHello()  
{  
    String message = "say hi";  
    println(message);  
}
```

```
void draw()  
{  
    String message = "Hey there!";  
    sayHello();  
    println(message);  
}
```

variables in different scopes can have the same name, but they do not share data – they are completely separate.

vocab. variables within a function are called **local variables**

function parameters

Reusing code...

`drawRandomSquares` – great!

what if we want variation?

draw different sizes?

draw different colors?

We need to give parameters to the function.

Send data to a function-> parameters

instead of

```
drawRandomSquares();
```

You want to do:

```
drawRandomSquares(20); // max size
```

Or

```
drawRandomSquares(20, 255); // max size, color
```

the numbers are parameters

Send data to a function-> parameters

void functionName (parameterType parameterName)

You can add parameters to the function

```
void drawRandomSquares() {
```

```
..
```

```
}
```

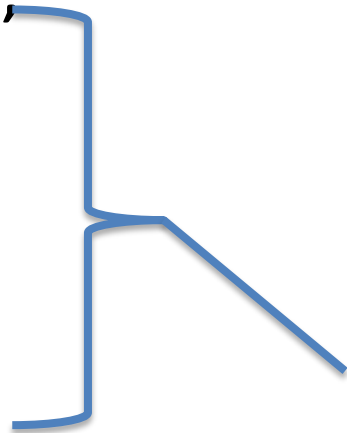
```
void drawRandomSquares(int size) {
```

```
..
```

```
}
```

user-defined functions: syntax

```
void methodName (parameterType parameterName)  
{  
    ...//processing code;  
}
```



function
body



function
header

How to use parameters inside a function?

```
void drawRandomSquares(int size) {  
    println(size);  
}
```

the parameters become local variables

data from outside is **copied** into the size variable and you can use it within the function just as any other variable.


Update drawRandomSquares

function parameters and... data is always copied

```
void changeData(int data) {  
    data = data * 2;  
}
```

the information is copied here. inside the function you only have a copy. changes here do not reflect back!

```
void draw()  
{  
    int data = 4;  
    changeData(data);  
    println(data);  
}
```



Variant - keep tunnel vision!!!!!!

```
void changeData(int data) {  
    data = data * 2;  
}
```

this variable name "data"
is important for inside the function
only. not related to how function is used

```
void draw()  
{  
    int number = 4;  
    changeData(number);  
    println(number);  
}
```

because information is copied, the
"caller" can give a literal, a variable,
do calculations, etc... NAME is irrelevant

another example...

make a function called `statusPrint` which prints a `String` message surrounded by `--` and with its length appended in parenthesis. Print it in the bottom left corner

: e.g.,

if you call:

```
statusprint ("Hi!");
```

it will print:

```
-Hi!- (3)
```

multiple parameters

a function can have as many parameters as you like, and they can be of different types!!

Let's update drawRandomSquare to also take the color

multiple parameters – just use commas.

```
void drawRandomSquare(int maxSize, int color) {  
...  
}
```

you can mix types, e.g., have a double, a String, etc.

you can have as many parameters as you want, usually $\leq \sim 3$

Another example...

Let's make a program with squares moving randomly around the screen – bad guys!

1 bad guy:

Global variables

finals: max move size (20), bg color

bad guy color, size, x and y

Make random moves

Separate x and y

- can move left or right by max move
- so range of movement is $-\text{max} \dots \text{max}$
- If 20, then 40 possible positions (41?)
- how to generate this as a random number?
 - $\text{random}(2 * \text{max}) - \text{max} \rightarrow -\text{max} \dots \text{max}$

See the repetition? Hmm.

Make sure doesn't move off the edges
of the screen

Use ifs, or min and max

Draw the badguy – set the stroke and fill colors,
too

Scale to three bad guys!!!

What a mess...

Use functions to simplify and reduce repeated code

New function!

```
drawBadGuy(x, y, size, color);
```

Draw block is now a little simpler... but still a lot of code being repeated..

Can we make a command like..

```
moveBadGuy(x, y, xmin, xmax, ymin, ymax)
```

```
moveBadGuy(badGuy1X, badGuy1Y, 0, width-1,  
0, height-1)
```

No – since data is only copied in, any changes that happen in that function are not reflected back in our variables. It is thrown away.

What functions give us data back?
What do they look like?

max, min, random...

```
int result = max(10,4);
```

Functions can only return one piece of data

They can give you an integer,

A float

A string

Etc.

Send data back from a function:

let's make a function `myMax(int a, int b)` which gives us an integer to represent the largest of the two:

```
int bigger = myMax(5, 2); // expect 5 to be the answer
```

first – let's implement this using the tools we already have

we can calculate important information – but how do we send it back?

Send data back from a function:

```
returnType functionName (parameterType parameterName)
```

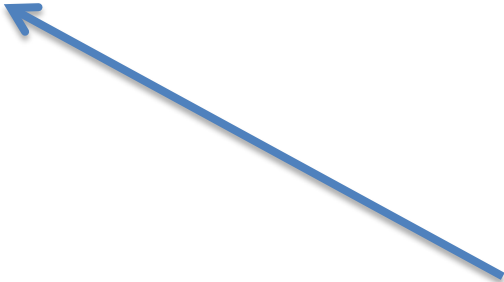
```
int myMax(int a, int b) {  
    int result = a;  
    if (b>a)  
        result = b;  
    return result;  
}
```

the **return** command does two things:

- it ends the function and returns to where it was called from
- it passes data along from the function to the caller

user-defined functions: syntax

```
int myMax(int a, int b) {  
    int result = a;  
    if (b>a)  
        result = b;  
    return result;  
}
```




these are “local variables”, only exists within the function. this name is not related to how you can use the function

```
...  
int max = myMax(10,20);
```

user-defined functions: syntax

```
int myMax(int a, int b) {  
    int result = a;  
    if (b>a)  
        result = b;  
    return result;  
}  
  
...  
int max = myMax(10,20);
```

A blue arrow originates from the closing curly brace of the `myMax` function definition and points downwards to the function call `myMax(10,20)` in the subsequent code line, illustrating the relationship between the function's definition and its invocation.