Electronic Prototyping

Functions

Lesson 4

Functions

Segmenting code into functions allows a programmer to create modular pieces of code that perform a defined task and then return to the area of code from which the function was "called".

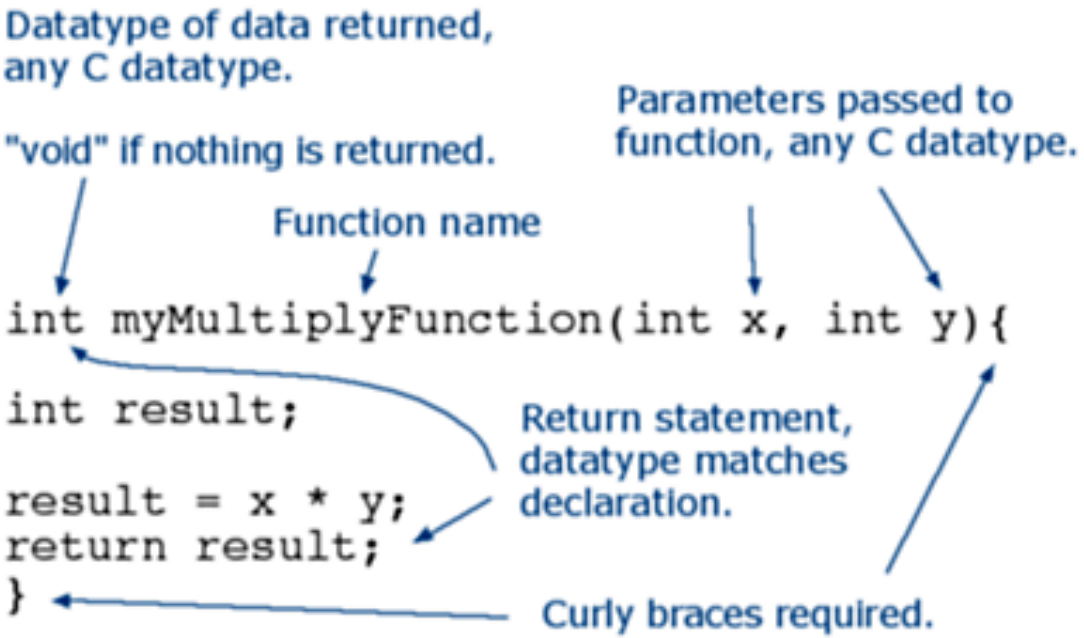
The typical case for creating a function is when one needs to perform the same action multiple times in a program.

Advantages

- Functions help the programmer stay organized. Often this helps to conceptualize the program.
- Functions codify one action in one place so that the function only has to be thought out and debugged once.
- This also reduces chances for errors in modification, if the code needs to be changed.
- Functions make the whole sketch smaller and more compact because sections of code are reused many times.
- They make it easier to reuse code in other programs by making it more modular, and as a nice side effect, using functions also often makes the code more readable.

There are two required functions in an Arduino sketch, setup() and loop(). Other functions must be created outside the brackets of those two functions.

How to create a function



A function must have a return type. If the function does not return anything, it has a return type of void.

How to call a function

```
void loop(){
int i = 2;
int j = 3;
int k;

k = myMultiplyFunction(i, j); // k now contains 6
}
```

Our function needs to be *declared* outside any other function, so "myMultiplyFunction()" can go either above or below the "loop()" function.

Scope

- A **global** variable is one that can be seen by every function in a program.
- **Local** variables are only visible to the function in which they are declared.
- In the Arduino environment, any variable declared outside of a function (e.g. setup(), loop(), etc.), is a global variable.

Scope

Example Code

```
int gPWMval;  // any function will see this variable

void setup() {
  // ...
}

void loop() {
  int i;    // "i" is only "visible" inside of "loop"
  float f;  // "f" is only "visible" inside of "loop"
  // ...

  for (int j = 0; j < 100; j++) {
    // variable j can only be accessed inside the for-loop brackets
  }
}
```

Example: rainbow with a RGB led

RGB led

- Create a function to obtain different colors by assigning a random value (0-255) to RED, GREEN and BLUE pins

Interrupts

- An interrupt, in microcontroller context, is a signal that temporarily stops what the CPU is currently working at.
- When a sketch is executed, the top most lines are run first. So logically the *setup()* function is run before the *loop()* function. The *loop()*function is an endless loop so there is no way to exit it.
- If we will now use interrupts, we add a third function named *isr()*. ISR is short for *Interrupt Service Routine*. This is where the program jumps to whenever there is an interrupt. An ISR cannot have any parameters, and they shouldn't return anything.

Interrupts

When does the program jump to isr()? For the Arduino platform, there will be an interrupt when specific pins change their state. If the interrupt pin is normally high, when it becomes low, then the interrupt is triggered and the program jumps to isr().

```
1  void setup(){
2  }
3
4  void loop(){
5  }
6
7  void isr(){
8  }
```

Interrupt pins for different Arduino boards

BOARD	DIGITAL PINS USABLE FOR INTERRUPTS
Uno, Nano, Mini, other 328-based	**2, 3**
Uno WiFi Rev.2	all digital pins
Mega, Mega2560, MegaADK	2, 3, 18, 19, 20, 21
Micro, Leonardo, other 32u4-based	0, 1, 2, 3, 7
Zero	all digital pins, except 4
MKR Family boards	0, 1, 4, 5, 6, 7, 8, 9, A1, A2
Due	**all digital pins**
101	all digital pins (Only pins 2, 5, 7, 8, 10, 11, 12, 13 work with **CHANGE**)

Syntax: attachInterrupt()

- **attachInterrupt**(digitalPinToInterrupt(pin), ISR, mode); Arduino Uno
- **attachInterrupt**(pin, ISR, mode); Arduino Due

Syntax: attachInterrupt()

mode: defines when the interrupt should be triggered. Four constants are predefined as valid values:

- **LOW** to trigger the interrupt whenever the pin is low,
- **CHANGE** to trigger the interrupt whenever the pin changes value
- **RISING** to trigger when the pin goes from low to high,
- **FALLING** for when the pin goes from high to low.

The Due, Zero and MKR1000 boards allows also:

- **HIGH** to trigger the interrupt whenever the pin is high.

Example

```
const byte ledPin = 13;
const byte interruptPin = 2;
volatile byte state = LOW;

void setup() {
pinMode(ledPin, OUTPUT);
pinMode(interruptPin, INPUT_PULLUP);
attachInterrupt(digitalPinToInterrupt(interruptPin), blink, CHANGE);   //Arduino Uno
attachInterrupt(2, blink, CHANGE);   //Arduino Due
}
void loop() {
digitalWrite(ledPin, state);
}
void blink() {
state = !state;
}
```