Function Basics
Functions

• Functions are not strictly necessary for a programming language to be complete (in the Turing complete sense)
• But they are useful!
  • And implementing some things without functions may require basically rebuilding the machinery of functions anyway
• What makes functions useful?
  • Encapsulation of code — reuse functionality
  • Encapsulation of state — local variables
  • Recursion

```c
int main() {
    return p(2) + p(3) + fact(3);
}

int p(int x) {
    return x * x;
}

int fact(int x) {
    int (x == 0) return 1;
    return x * fact(x - 1);
}
```
Encapsulation of code

- Functions are subroutines
  - **Call** a function: execute routine then return back to where you called it from
  - Need a **function address** to figure out where the routine’s code is
  - Need a **return address** to figure out where to return to
  - These are addresses of code, not data
  - The function **making the call** is the **caller**; the function **being called** is the **callee**

```c
int foo() {
    int x;
    x = 2;
    print(bar(x));
    print(bar(x + 1));
    return 0;
}

int bar(int y) {
    int x;
    print(y);
    x = y * y;
    return x;
}
```
Encapsulation of code

• Functions can modify their behavior based on how they are called
  • Pass a different set of arguments to the function, perform a different computation
  • Need some way of **binding** the **arguments** to a function to the **parameters** of a function
  • Need some way of **passing** data between caller and callee

```c
int foo() {
    int x;
    x = 2;
    print(bar(x));
    print(bar(x + 1));
    return 0;
}

int bar(int y) {
    int x;
    print(y);
    x = y * y;
    return x;
}
```
Encapsulation of data

• Local variables in a function are not visible outside the function
  • Modification to local variables are not seen anywhere else
  • Local variables retain their value even after calling a function and returning from it
• Need a place to store local variable on a per-function basis
  • New local storage each time a function is called: a frame or activation record
  • Local storage persists until a function returns: a stack

```c
int foo() {
    int x;
    x = 2;
    print(bar(x));
    print(bar(x + 1));
    return 0;
}

int bar(int y) {
    int x;
    print(y);
    x = y * y;
    return x;
}
```
Recursion

• Once you have the ability to call a function multiple times, with different parameters each time, and local storage, you can do **recursion**
• The basis of many models of computation

```c
int fib(int x) {
    int s1;
    int s2;
    if (x < 2) {
        return 1;
    }
    s1 = fib(x - 1);
    s2 = fib(x - 2);
    return s1 + s2;
}
```