Functions

What are functions good for?

- **Help avoid code repetition**
  - parameterization of functional blocks, encapsulation of code
- **Scalability**: build large and robust software systems
  - modular, top-down design, unit testing of code modules
- **Portability and re-use**

Terminology

- **Subroutine/Procedure**: A call/return block of code that does not return a value, but may take arguments and modify their value.
- **Function**: A call/return block of code that returns a value. Functions may take arguments and modify their value.
- **Method**: A function or subroutine that is part of a class.
- **Argument**: An expression used when calling a function or subroutine.
- **Parameter**: The identifier declared in a function or subroutine definition that will connect to the corresponding argument in a function call.

Parameter Passing

- **Pass by value**: the computer evaluates the argument expression and places a copy of its value into the memory address referenced by the parameter. (C, C++, Java)
- **Pass by reference**: the computer evaluates the argument expression and places a reference to the result into the memory address referenced by the parameters. (C++)

Show Swap.java, ask the output for swap(int, int) [won’t swap a and b]
ask the output for swap(Integer, Integer) [won’t swap a and b]
ask the output for resetVal (Item item) [reset the item val]
Why? [Java is pass by value]
```java
public class Swap {
    public static void swap (int a, int b) {
        int t = a;
        a = b;
        b = a;
    }

    // If change the reference to refer to another location, 
    // any changes to the reference are not reflected 
    // back in the main function 
    public static void swap (Integer a, Integer b) {
        Integer t = a;
        a = b;
        b = a;
    }

    // If change the member of an object referred by the reference
    // the changes are reflected back, as even it's a copy of
    // the object's reference, it refers to the same object
    public static void resetVal (Item item) {
        item.val = 0;
    }

    public static void main (String[] args) {
        int a = 5;
        int b = 10;
        System.out.println("Before swapping a = " + a + " b = " + b);
        swap(a, b);
        System.out.println("After swapping a = " + a + " b = " + b);

        Integer i = 5;
        Integer j = 10;
        System.out.println("Before swapping i = " + i + " j = " + j);
        swap(i, j);
        System.out.println("After swapping i = " + i + " j = " + j);

        Item t = new Item(5);
        System.out.println("Before resetting t.val = " + t.val);
        resetVal(t);
        System.out.println("After resetting t.val = " + t.val);
    }
}

class Item {
    int val;

    public Item (int v) {
        val = v;
    }
}
```

```
mbp-190250:Functions yingli$ javac Swap.java
mbp-190250:Functions yingli$ java Swap
Before swapping a = 5 b = 10
After swapping a = 5 b = 10
Before swapping i = 5 j = 10
After swapping i = 5 j = 10
Before resetting t.val = 5
After resetting t.val = 0
mbp-190250:Functions yingli$
```
• Now we know that Java is pass-by-value. How about C?
  - Using swap function is a good way To check whether a language uses pass-by-value or pass-by-reference. The the change happened inside the swap function cannot reflect to the calling function, the language is pass-by-value.
  - Show swapf.c, and ask the output and the parameter passing of C. [pass by value]

```c
#include <stdio.h>

int swap_f (int x, int y) {
    int t;
    t = x;
    x = y;
    y = t;

    return 0;
}

int main () {
    int a = 3, b = 5;
    printf("Before swap a = %d, b = %d \n", a, b);
    swap_f(a, b);
    printf("After swap a = %d, b = %d \n", a, b);

    return 0;
}
```

• Although C uses passing by value, we have to use pointer to simulate passing by reference.

```c
#include <stdio.h>

int swap (int *x, int *y) {
    int t;
    t = *x;
    *x = *y;
    *y = t;

    return 0;
}

int main () {
    int a = 3, b = 5;
    printf("Before swap a = %d, b = %d \n", a, b);
    swap(&a, &b);
    printf("After swap a = %d, b = %d \n", a, b);

    return 0;
}
```

However, passing pointers as values is not equivalent to pass-by-reference. We can tell the difference after knowing what is pass-by-reference in C++.

• C++ allows passing by reference. Show swap.cc.
The above swap function passes arguments as references. Using pass-by-reference, the only thing the swap function can do is update the values in the memory referred by the references. It’s syntactically incorrect to change the reference values inside the swap function. This is to say it’s invalid to do anything like `&a = &b` in the above swap function.

Note: passing by reference is not equivalent to passing pointer by value.
- If a function is pass-by-reference, it’s unlikely to modify the reference itself in the function be the value referred by the reference.
- If a function is pass-by-value and the value is a pointer, it’s possible to reassign another address to the pointer, and the changes on the value referred by the new address won’t impact the value stored in the original address.

```cpp
#include <iostream>
using namespace std;

int swap (int &a, int &b) {
    int tmp = a;
    a = b;
    b = tmp;
    return 0;
}

int main () {
    int a = 3, b = 5;
    cout << "Before swap a = " << a << " b = " << b << endl;
    swap(a, b);
    cout << "After swap a = " << a << " b = " << b << endl;
    return 0;
}
```

Before f, a = 1
After f, a = 3
Before g, a = 3
After g, a = 3
Another thing you may want to pay attention to pass-by-reference is the aliasing, which means reference to the same entity by different names.

```cpp
#include <iostream>
using namespace std;

int open_to_problems (int &a, int &b) {
    a = a + 1;
    b = b + 1;
    return 0;
}

int main () {
    int x = 10, y = 20;
    cout << "Before open_to_problems, x = " << x << " y = " << y << endl;
    open_to_problems( x, y );
    cout << "After open_to_problems, x = " << x << " y = " << y << endl;

    cout << "Before open_to_problems, x = " << x << endl;
    open_to_problems( x, x );
    cout << "After open_to_problems, x = " << x << endl;

    return 0;
}
```

Before open_to_problems, x = 10, y = 20
After open_to_problems, x = 11, y = 21
Before open_to_problems, x = 11
After open_to_problems, x = 13