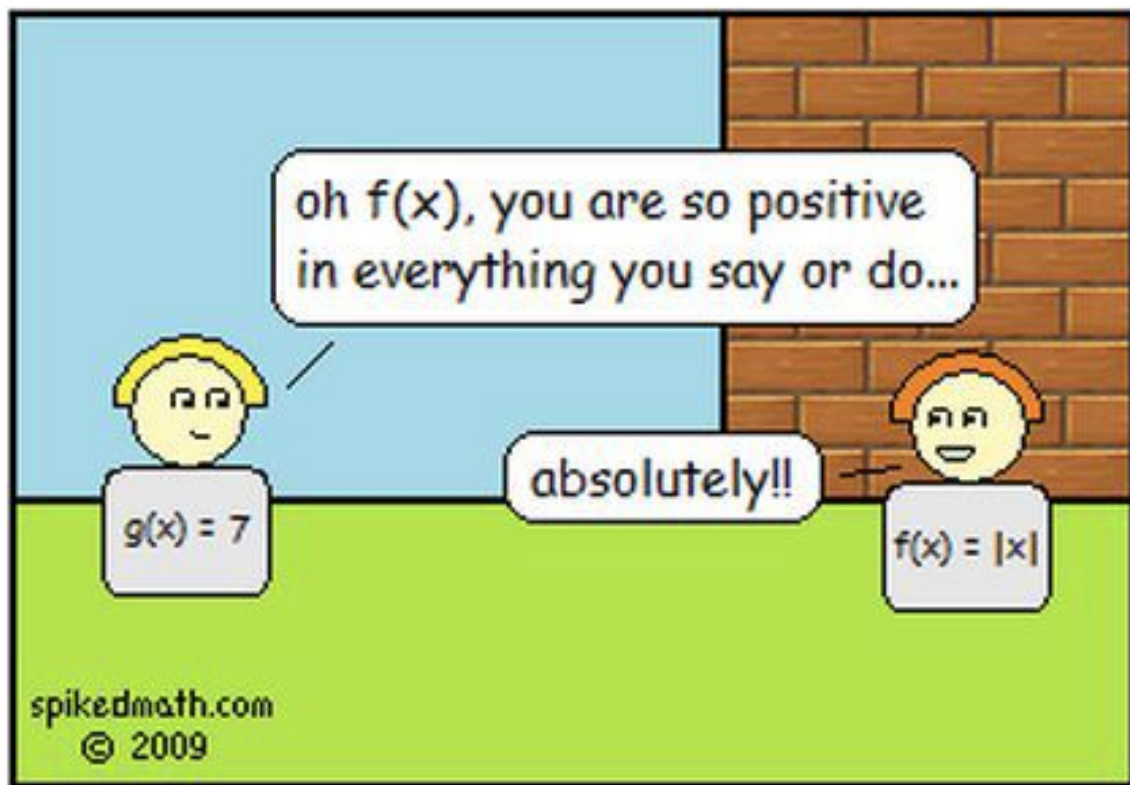


# Functions



# Due this week

- **Homework 2**

- Write solutions in VSCode and paste in Autograder, **Homework 2 CodeRunner**. Check the due date! **No late submissions!!**

- **Homework 2 Quiz**

# Topics for today

- Introducing functions in C++
- Understanding parts of a function
- How do we think about writing them..
- Examples... (let's switch to VS Code)



# Functions

- It's a set of lines of code with a name, and a defined sequence of actions.
- It encapsulates (combines) a collection of instructions that go together to accomplish a task.

```
void greetUser(string name) {  
    cout << "Hello " << name << endl;  
}  
|
```

# Functions as a blackbox

- You can think of a function as a “black box”
  - Know what the box does, but can’t see what’s inside
  - Like a pressure cooker -- can’t see inside, know what it does



# Parts of a function

- Let's breakdown some components of the snippet below

```
void greetUser(string name) {  
    cout << "Hello " << name << endl;  
}
```

# Parts of a function

- Function Name - The unique name you provide to the function (just like how you name a variable)
- Function Parameters - The input/s to the function.
- Function Definition - This is the sequence of instructions (code) that the function does to accomplish something
- Function Return Value - The output we get from a function

```
int getPriceOfCar(string make, char model, int year, double mileageDriven) {  
    int price_of_car = 0;  
  
    /* Some logic here to set the price of the car depending on above factors  
    **  
    **  
    **  
    */  
  
    return price_of_car;  
}
```

# Which of the following are some examples of functions

- `if() {.....}`
- `#include`
- `int main() {}`
- `using namespace std;`
- `setprecision(5)`
- `float`
- `x % y;`



# Steps to write your own function

- Let's suppose you need to determine if a number is an odd or an even.
  - Understand the task at hand - This helps you get clarity of the function you need to write, such as its name and its body/definition. Ex: `isNumberEven()`, `isOddNumber()`
  - Think about what inputs it needs! - This helps you determining the function arguments (also called formal parameters). Ex: `isNumberEven(int number)`, `isOddNumber(int num)`
  - Think about what output it needs to generate - This helps in determining the return type and value from the function. Ex: `bool isNumberEven(int number)`, `void isOddNumber(int num)`.
- What you did so far - You wrote the function definition!

**visual studio code**



# Calling a function

- A function is called to execute by calling its name. You can call the function anywhere in a program.
- Let's take the example of isNumberEven.

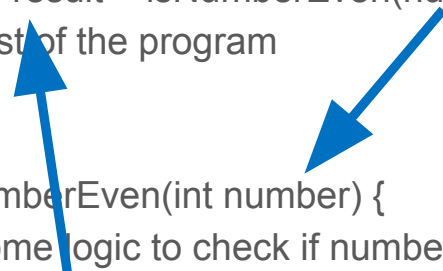
```
int main() {  
    int x = 341;  
    bool result = isNumberEven(x);  
}
```

- Remember
  - If the function needs input/s, provide it at the time of calling. (arguments)
  - Pass the correct data type to the function
  - If the function returns something, store that in a variable (or print it).
  - Notice that this function can be called any number of times you wish to! Reusable and modularity benefits!

# Calling a function

- Parameter Passing

```
int main() {  
    int number = 324;  
    bool result = isNumberEven(number);  
    // rest of the program  
}  
  
bool isNumberEven(int number) {  
    // some logic to check if number is even or not  
    return result; // return true or false  
}
```



The diagram consists of two blue arrows. One arrow originates from the parameter 'number' in the function call 'isNumberEven(number)' within the 'main' function and points to the parameter 'number' in the function definition 'bool isNumberEven(int number)'. The second arrow originates from the variable 'result' in the assignment 'bool result = isNumberEven(number);' and points to the 'return result;' statement in the 'isNumberEven' function definition.

# Flow of execution during function call

- Main function creates a variable called number with some initial value.
- It calls the function isNumberEven and provides the number as input
- The main function execution stops, and isNumberEven function starts!
- Logic is executed and a value is returned to the calling function (the main function)
- Now main function continues its execution

## Another example - Adding two numbers

```
int main() {  
    int a = 5, b = 7;  
    int sum = getSum(a, b);  
    cout << "Sum = " << sum << endl;  
    return 0;  
}  
  
int getSum(int a, int b) {  
    int sum = a + b;  
    return sum;  
}
```

# Functions - Declaration and Definitions

# Function Declarations and Definitions

What's wrong with the code below?

```
int main() {  
    int a = 5, b = 7;  
    int sum = getSum(a, b);  
    cout << "Sum = " << sum << endl;  
    return 0;  
}  
  
int getSum(int a, int b) {  
    int sum = a + b;  
    return sum;  
}
```



# Function Declarations (Prototype Statements)

---

- It is a compile-time error to call a function that the compiler does not know
  - just like using an undefined variable.
- So define all functions before they are first used
  - But sometimes that is not possible, such as when 2 functions call each other

# Function Declarations (Prototype Statements)

---

- Therefore, some programmers prefer to include a definition, aka "prototype" for each function at the top of the program, and write the complete function after `main() {}`

- A prototype is just the function header line followed by a semicolon:

```
int getSum(int a, int b);
```

- The variable names are optional, so you could also write it as:

```
int getSum(int) ;
```

# Correct solution - Add Function Declaration

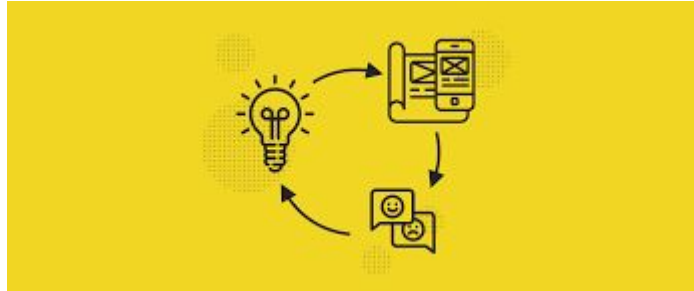
---

```
int getSum(int, int);  
  
int main() {  
    int a = 5, b = 7;  
    int sum = getSum(a, b);  
    cout << "Sum = " << sum << endl;  
    return 0;  
}  
  
int getSum(int a, int b) {  
    int sum = a + b;  
    return sum;  
}
```



---

# Let's do some prototyping



# Examples

---

- It's your birthday today and you want to get a free cup of coffee at Starbucks! Write a function that returns true if the current date is the same as your birthday.



# Examples

---

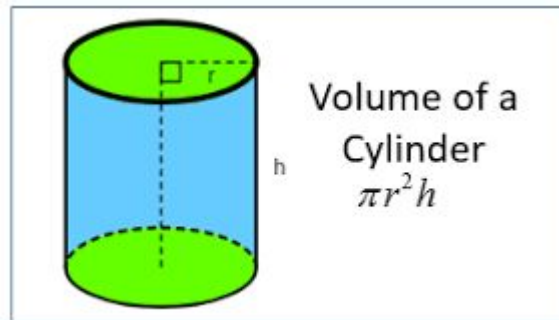
- You've decided to buy a house. Write a function that helps you calculate the cost of a house, depending on factors such as square footage, number of bedrooms, location and whether it's got a swimming pool or not!



# Examples

---

- Write a function to print the volume of a cylinder. Don't return anything, but rather just print the volume





# Examples

---

- You're designing a software like Google Calendar. Write a function that takes in the month ("Jan", "Feb", ... , "Dec"), date, and the year and returns the day of the week ("Sun", "Mon", ... , "Sat").



# Pitfalls to avoid

- Arguments
  - Providing incorrect number of arguments, or incorrect data type for arguments -> **Compile time ERROR**
- Return values
  - Returning incorrect data type for return value -> **Compile time ERROR**
  - Missing a return statement -> **Compile time ERROR**
- Calling a function before declaring it! -> **Compile time ERROR** (Use function prototyping to solve it, or just declare it above where it's called)



# Some Tips while writing functions

- Always write comments - what the function does, what parameters it takes, and what it returns
- Don't modify Parameter variables! The behavior is more complex.

```
int withdraw(int amount, int balance) {  
    balance = balance - amount;  
}
```



- Remember, cout statement printed values to the screen, while return statements from functions provides output to the calling function
- Keep the size of function short (usually one screen long)
- Test the function multiple times, to see if it works as expected with various inputs