Functions

Math world: $f: \mathbb{R} \to \mathbb{R}$

- Maps each input to a single output
- Might be multivariate, $f(x,y)$
  (domain is $\mathbb{R} \times \mathbb{R}$)

In C/C++, they work a bit differently.

Notation:

Math-world: $f: \mathbb{Z} \to \mathbb{R}$	"prototype" C/C++: `double f( int );` range domain
f is "single-valued" i.e., $f(2)$ can never change	$f(2)$ might give different results each time it is evaluated!

defined by rules: $f(x) = x^2+1$	`double f(int x)`
	`{`
	`    return x*x +1;`
	`}`

Example usage: "function call"

```
int main()
{   double y = f(4);
    cout << y << "\n";
    // print 17
    return 0;
}
```

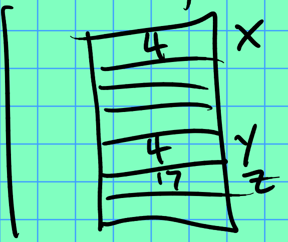
"By value" vs "By reference" params

say f is defined as

```
int f(int x) { return x*x+1; }
```

Now in another function, we have this:

```
// in main...
    int y, z;
      y = 4;
    z = f(y);
```



What does memory look like during the call?
What's the (physical) relationship between x, y?

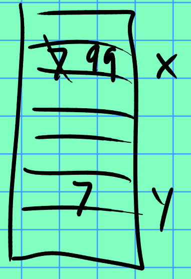
Example:

```
void g(int x)            | void h(int& x)
{       x = 99;          | {
        return;          |     x= 99;
                         |     return;
}                        | }
```

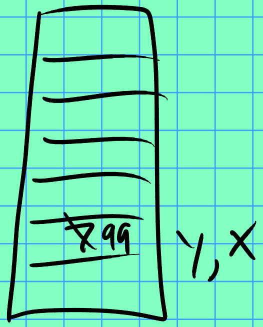
```
// in main
    int y = 7;
    g(y);
     cout << y << "\n";  // prints 7
    h(y);
    cout << y << "\n";  // prints 99
```

Picture for calls to g vs h:

g:



h:



"by value"

No "&"

"by reference"

with "&"

```
int readint()
{   int n;
    cin >> n;
    return n;
}
// in main
  int m = readint();
```