Functions

- So far, have seen numbers, assignments, input/output, comparisons, looping constructs
  - Sufficient to be **Turing Complete**
- But code lacks abstraction
  - Have to reload file every time want to use
  - Can't use same variable names in other pieces of code
  - Can quickly get cumbersome to read and maintain
- Functions give us abstraction – allow us to capture computation and treat as if primitive

Compute anything

A simple example

- Suppose we want to z to be the maximum of two numbers, x and y
- A simple script would be

```
if x > y:
    z = x
else:
    z = y
```

Have to copy everywhere…
Can't reuse x, y!

Capturing computation as a function

- Idea is to encapsulate this computation within a scope such that can treat as primitive
  - Use by simply calling name, and providing input
  - Internal details hidden from users

Black box

- Syntax

```
def <function name> (<formal parameters>):
    <function body>
```

- def is a keyword
- Name is any legal Python name
- Within parenthesis are zero or more formal parameters – each is a variable name to be used inside function body

A simple example

```
def max(x, y):
    if x > y:
        return x
    else:
        return y
```

- We can then invoke by

```
z = max(3, 4)
```

- When we call or invoke max(3, 4), x is bound to 3, y is bound to 4, and then body expression(s) are evaluated

Function returns

- Body can consist of any number of legal Python expressions
- Expressions are evaluated until
  - Run out of expressions, in which case special value None is returned
  - Or until special keyword return is reached, in which case subsequent expression is evaluated and that value is returned as value of function call

Summary of function call

1. Expressions for each parameter are evaluated, bound to formal parameter names of function
2. Control transfers to first expression in body of function
3. Body expressions executed until return keyword reached (returning value of next expression) or run out of expressions (returning None)
4. Invocation is bound to the returned value
5. Control transfers to next piece of code