Functions
Encapsulating Logic
Functions

Groups of statements that can be run more than once in a program

Reusable chunks of code

Take input, provide output

Can be reused in a variety of contexts

Maximize code reuse, minimize code redundancy

Encapsulate logic, splitting complex systems into manageable parts
**Calling Functions**

\[ \text{type}(3.7) \]

**Function name**

**Arguments passed into function**

**Return Value**

\( () \) tells Python to execute the function

Even if a function takes no input, the brackets are still required

Some functions do not return a value
Defining Functions

Just like writing a Python program but with some extra syntax

Function header:

• Begins with `def`
• Followed by name of function
• Function parameter list
• Ends with a `:`
Documentation String

Doc String
Optionally follows function header
Explains what function will do
May include example(s)
Almost always good to include for clarity and as a reminder
Function Body

All the code that follows the header

Simply an indented block of code with necessary statements

This code can use the variables from the function header

Function should return a value with the keyword `return`

After `return`, Python jumps out of the function and back to the program
**Function Syntax**

```python
def name(parameters):
    statements
    return None
```

A return value is not required.

Anything besides `return` are called "side effects".

A `print()` statement is an example of a side effect.
a
Main Function

main()

It is both common and a good idea to use a main function in your programs. This is usually the starting point of a program and is run by typing: `main()`.

This simplifies rerunning programs and as well as passing input values.
Function Parameters

Used to pass input into a function

Python passes values by reference

Default values can be set for parameters

```python
def name(parameter = 'default')
```
Variables

Local and Global

Variable scope is something we must pay attention to when using functions.

When and where is a variable accessible to your program?

Local variables are only usable within the function they are local to.

Global variables are usable by any function or code within your program.
Local Variables

def area(rad):
    import math
    return math.pi * rad ** 2

When this function ends, rad is automatically deleted because it is local.
Global Variables

```python
rad = 5

def area():
    import math
    return math.pi * rad ** 2

def change_rad(new_rad):
    rad = new_rad

This program will not allow `rad` to be changed because it is global

To change the radius variable:
```
```
Modules

A module is a group of related functions
Different from a regular Python program in that it acts like a toolbox
A module usually does not have a main() function
To use a module, simply import it
Importing Modules and Calling Functions

Calling function after importing module:

```python
import module_name

module_name.function()
```

Calling function after importing module along with all of its functions:

```python
from module_name import *

call_function()
```
Module Namespaces

Functions within a module are available when you import them.

Modules form namespaces.

Different modules with the same function name will not clash in the same program.

`module1.function(parameter)`

`module2.function(parameter)`

The only time functions may conflict is when you import all using `*`.
<table>
<thead>
<tr>
<th>Introduction to Computer Programming</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSCI-UA 2</td>
<td>Encapsulating Logic</td>
</tr>
</tbody>
</table>