Functions!!!

- Why functions? Functions provide a good way to design systems!
Coding Design!

● DRY principle - Don't Repeat Yourself!
  ● Loops help with this!
  ● Functions/procedures/modules help even more!
● Modular design
  ● Functions/procedures/modules help with this!
Functions!!!

• What's a function?!
  • we've already been using lots of functions:
    - print()
    - int(), str(), float()
    - math.sqrt()
    - help()
Functions!!!

- What's a function!?
  - a function is a stored set of instructions that perform a specific task
  - We can 'define' new functions
  - Let's see an example!
def welcome():
    '''print an intro to our program'''
    print('Welcome user!')
    print("We're learning functions!")
    print('Tomorrow... who knows!')
Functions!!!

How do we use functions?
def welcome():
    '''prints an intro to our program'''
    print('Welcome user!')
    print("We're learning functions!")
    print('Tomorrow... who knows!')

# call the welcome function
welcome()
def welcome():
    '''prints an intro to our program'''
    print('Welcome user!')
    print('We're learning functions!')
    print('Tomorrow... who knows!')

    ''' main program logic ''
welcome()
# other program logic
Making programs more modular

def welcome():
    '''prints an intro to our program'''
    print('Welcome user!')
    print('We're learning functions!')
    print('Tomorrow... who knows!')

def main():
    ''' main program logic '''
    welcome()
    # other program logic

main()
Functions!!!

• comprised of:
  • name - name used to call the function
  • parameters (optional) - variables passed to the function
    – In addition to parameters being optional themselves, there are two types of parameters:
      • required - parameters that a function requires
      • optional - some parameters can be optional parameters (e.g. print's 'sep' and 'end' parameters)
  • code - statements that run on given parameters
  • return value (optional) - result returned by the function
Variable Scope

- Scope is the context in which a variable is accessible (what code is the variable accessible in)
- Two main types of variables and their scope:
  - Global variables
  - Local variables
Global Variables

- Global variables - variables declared outside of a function or class that are accessible by the main program and other functions in the main program.
  - These variables are accessible globally and modifiable globally (from functions)
  - If accessed in a function you need to use the keyword 'global' when first declaring it so that python knows you're trying to use a global variable (otherwise it defaults to a local variable)
  - Generally a bad idea to use globals
  - These are what we have been using (so far)
Local Variables

- Local variables - variables that are defined inside a specific code context and are not accessible outside of that context. In this case inside of a function.
  
  - In functions we sometimes call these 'function variables'
  
  - Deleted after function is done running
  
  - Using local variables instead of global variables ensures good modular design
    
    - in general, a function shouldn't be dependent on code outside of itself, it should be self contained.
Parameters

Parameters are variables that are passed to a function.

- Local to the function they are passed to (not accessible outside the function)
- Changing the contents of a parameter does not change the contents of a original variable passed to that function.
- Lets see an example.
def welcome(topic):
    '''prints an intro to our program
    param - topic is printed as part of the intro
    '''
    print('Welcome user!')
    print("We're learning about", topic)
    print('Tomorrow... who knows!')
Function - Optional Parameters

def welcome(topic='functions'):
    '''prints an intro to our program
    param - topic is printed as part
    of the intro
    '''
    print('Welcome user!')
    print("We're learning about", topic)
    print('Tomorrow... who knows!')
def welcome(topic, print_tomorrow=True):
    '''prints an intro to our program'''
    print('Welcome user!')
    print("We're learning about", topic)
    if print_tomorrow:
        print('Tomorrow... who knows!')
def addition(x, y):
    '''Does simple addition
    returns - addition of x and y'''
    return x+y
def addition(x, y):
    '''Does simple addition
    returns - addition of x and y'''
    return x+y

def main():
    print('3 plus 4 equals', addition(3, 4))

main()
Functions with Keywords

- When calling a function, we can pass parameters using keywords instead of passing them in the same order as the function definition.

- Example:
  ```python
def volume_cone(radius, height):
    # CODE GOES HERE

    # all of these function calls work and are equivalent
    volume_cone(2,3)  # using default ordering
    volume_cone(radius='2', height='3')  # using keywords
    volume_cone(height='3', radius='2')  # using keywords
  ```
Functions with Keywords

- When calling a function, we can pass parameters using keywords instead of passing them in the same order as the function definition

- **Pro:**
  - You don't have to remember the order of parameters as long as you remember the name of the parameters

- **Con:**
  - If the function parameter names change, all function calls will need to be changed (what if the parameter in the function definition was misspelled?)
Libraries/Modules

- A library is a set of utilities that provide functionality that apply to a specific topic (using functions).
  - math libraries - import math
  - python interpreter system libraries - import sys
  - operating system libraries - import os
  - random number libraries - import random
  - date related libraries - import date