Objectives

- Wrap up functions
  - Passing parameters
  - Default parameters
- Modules
- Testing functions

I want to

- Create a list of even numbers to 100
- Iterate through the elements of a list
- Iterate through the positions of a list
  - But look at the values in the list
- Iterate through the positions of a string
- Iterate through the characters of a string
- Do something to seach line in a file

DoND Advice

- Don't worry about the formatting of the display at first
  - Do a first pass implementation of all the functions, then debug and refine the functions

Review: Top-Down Design

- Alternative design approach
  1. Write code that calls functions
  2. Define functions
- Benefits
  - Design interface for functions first
    - How will function work? (i.e., the input parameters and what is returned)
  - Don't worry about implementation until it's clear what functions need to do

PASSING PARAMETERS

Passing Parameters

- Only **copies** of the actual parameters are given to the function
  - For **immutable** data types Which are?
- The *actual* parameters in the calling code do not change
- **Swap example:**
  - Swap two values in script
  - Then, put into a function

$$\boxed{\begin{matrix} x = 5 \\ y = 7 \end{matrix}} \Rightarrow \boxed{\begin{matrix} x = 7 \\ y = 5 \end{matrix}}$$

swap.py

Lists as Parameters to Functions

If a list that is passed as a parameter into a function is **modified *in* the function**, the list **is modified *outside* the function**

- Lists are **not** passed-by-value/copied
- Different from immutable types (e.g., numbers, strings)

- Parameter is actually a **pointer** to the list in memory

Problem: Sort a list of 3 numbers, in descending order

```
# order list such that list3[0] >= list3[1] >= list3[2]
def descendSort3Nums( list3 ):
```

Called as:

```
list = …
descendSort3Nums(list)
print(list)
```

How implemented with list methods?
Can we do this using only 3 comparisons?

Descend Sort a List w/ 3 elements

```
def descendSort3Nums(list3):
   if list3[1] > list3[0]:
       # swap 'em
       tmp = list3[0]
       list3[0] = list3[1]
       list3[1] = tmp

   if list3[2] > list3[1]:
       tmp = list3[1]
       list3[1] = list3[2]
       list3[2] = tmp

   if list3[1] > list3[0]:
       tmp = list3[0]
       list3[0] = list3[1]
       list3[1] = tmp
```

```
def main():
    list = [1,2,3]
    descendSort3Nums(list)
    print(list)
```

Function does **not** *return* anything. Simply modifies the `list3` parameter.

PARAMETER DEFAULTS

Defaults for Parameters

- Can assign a default value to a parameter
  - In general, in function header, default parameter(s) should come *after* all the parameters that *need* to be defined
- Example: **`range`** function
  - Didn't have to specify `start` or `increment` when calling the function
  - Default `start` = 0
  - Default `increment` = 1

Using Default Parameters

- By default, the `genWinningNum` function could assume that there are 4 numbers

Assigns a value to numNums **ONLY IF** not passed a parameter

```
def genWinningNum(numNums=4):
   winNum = ""
   for i in range(numNums):
      winNum += str(randint(MIN_VALUE,MAX_VALUE))
   return winNum
```

Examples of calling function:
```
genWinningNum(6)
genWinningNum()
genWinningNum(4)
```

CREATING MODULES

Where are Functions Defined?

- Functions can go inside of program script
  - Defined before use/called (if no `main()` function)
  - Or, below the `main()` function (preferred)
- Functions can go inside a separate **module**

Benefits of Defining Functions in Separate Module

- Reduces code in primary driver script
- Easier to reuse by importing from a module
- Maintains the "black box"
  - ***Abstraction***
- Isolates testing of function
- Write "test driver" scripts to test functions separately from use in script

Creating Modules

- Modules group together related functions and constants
- Unlike functions, no special keyword to define a module
  - A module is named by its filename

Just a Python file!

- Example, `oldmac.py`
  - In Python shell: `import oldmac`
  - Explain what happened

Defining Constants in Modules

- Constant in `oldmac.py`
  - `EIEIO`

Creating Modules

- So that our program doesn't execute when it is imported in a program, at bottom, add

```
if __name__ == '__main__' :
    main()
```

Not important how this works; just know when to use

- Then, to call `main` function
  - `oldmac.main()`
- Note the sub-directories now listed in the directory

Creating Modules

- Then, to call `main` function
  - `oldmac.main()`
- Why would you want to call a module's `main` function?
  - Automation
  - Use `main` function as driver to test functions in module
- To access one of the defined constants
  - `oldmac.EIEIO`

TESTING FUNCTIONS

Testing Functions

- Functions make it easier for us to test our code
- We can write code to test the functions
  - Input: parameters
  - Output: what is returned
    - We can verify programmatically

What are good tests for
`binaryToDecimal(binnum)` and `isBinary(candidate)`?

`binaryToDecimal.test.py`

Debugging Advice

- Build up your program in steps
  - Always write small pieces of code
  - Test, debug. **Repeat**
- Write function body as part of `main`, test
  - Then, separate out into its own function
  - Similar to process using in lab probs
- Test function separately from other code

Looking Ahead

- For Friday
  - Lab 7
  - Broader Issue: Digital Humanities
- Monday: Katherine Crowley talk at 7:30 p.m. in Stackhouse
  - Write in a response on Sakai for 10 pts extra credit
    - a one-sentence overview of the talk
    - the three most important points
    - most surprising thing she mentioned
    - at least one question that you wondered during the talk
    - how this talk relates to computer science
    - how computer science can help address one of the problems posed in the talk; explain a possible solution a little