CS 116
PRE-MIDTERM Q&A
Computations With \texttt{Str}, using \texttt{Int}, \texttt{Nat}, and \texttt{Float}

\begin{align*}
s &= \text{“CS”} \\
t &= \text{“116”} \\
x &= s + t => \\
r &= x - t => \\
y &= 3 * t => \\
z &= 4.5 * s =>
\end{align*}
Conditionals

if, elif, else use Boolean conditions to decide what parts of the code to execute – these are the tools we use to make decisions in Python.

Let’s trace through some function calls of `conditional_chaos(n)`

`conditional_chaos(10)`

`conditional_chaos(28)`

`conditional_chaos(25)`

```python
def conditional_chaos(n):
    ans = 0
    if n % 2 == 0:
        ans += 2
    if n % 4 == 0:
        ans += 2
    if n % 3 == 0:
        ans += 1
    if n % 5 == 0:
        ans += 5
    else:
        ans *= 3
    if n % 7 == 0:
        ans -= 2
    else:
        ans = ans**2
    return ans
```
Printing vs. Returning

**print**
- displays a value to the standard output
- does not stop the function’s execution
- does not return a value, and so cannot be stored in a variable

**return**
- return does not display values, except in the Wing interpreter
- stops the function’s execution
- returns a value which can then be processed further, or stored in a variable, etc.
Lists

- lists are data types which store multiple, connected pieces of data
- lists are mutable – individual entries can be changed without needing to reassign the entire list
- lists can be aliased – it is possible to have more than one variable referring to the exact same list in memory
Recursion

- We’ve got three major types of recursion:
  - structural
  - accumulative
  - generative
- We’ll take a look at some practice questions to get as comfortable as possible with recursive code.
Design Recipe

Suppose we want to write a function that will prompt the user for a command, perform that operation on a list, and then print a message when the task is completed. Let’s write the design recipe and some tests for this function, called `list_process`.

```python
def list_process(L):
    s = input("What would you like to do? ")
    if s.lower() == "append":
        new = input("What would you like to append? ")
        L.append(new)
    elif s.lower() == "pop":
        index = input("From where would you like to pop? " +
                        "Enter an integer between 0 and \{0\}. ".format(len(L) - 1))
        L.pop(int(index))
    elif s.lower() == "sort":
        L.sort()
    else:
        print("We don't recognize that command.")
        print("We only know 'append', 'pop', and 'sort'.")
    return list_process(L)
print("We have completed the command: '{0}'. Thank you!".format(s))
```
What is the result of calling the function: \(\text{whirly\_dirly}([1, 5, 1, 3, 7, 43], 3)\)?

def whirly_dirly(L, n):
    if n >= len(L):
        L.sort()
        return L[::2]
    elif L[n] > n:
        print("Whirly \{0\}".format(L[n]))
        x = input("Enter an integer: ")
        L[n] = n**int(x)
    elif L[n] < n:
        print("Dirly \{0\}".format(L[n]))
        L[n] = L[n] % n
    return(whirly_dirly(L, n+1))
Write a function `new_name` that consumes no parameters, but prompts the user for two strings, one containing a person's full current name, and the second containing the person's new last name. The function returns the person's updated name (which might be exactly the same).

For example: `new_name()`

user inputs: "Abigail Melissa Smith"
    "Jones"
=> "Abigail Melissa Jones"
new_name()
user inputs: "Timothy Howe"
    "Howe-Williams"
=> "Timothy Howe-Williams"
Write a function `sorted_copy` that consumes a list `L` of distinct integers, and returns a copy of `L` which is in sorted order, by using the following technique:
* find the smallest value in `L`
* create a list containing all entries except the smallest, and sort it
* combine the two pieces to create a copy of `L` in sorted order.

You may use abstract list functions and other list functions, except for `sort` and `sorted` :-)

For example, `sorted_copy([4, 3, 5, 2, 1])` => `[1, 2, 3, 4, 5]`
Write a function called `clean_list`, that takes two parameters: a list, \( L \), and another single parameter of any type, \( \text{target} \). \( \text{clean_list}(L, \text{target}) \) returns the number of times that \( \text{target} \) appears in the list \( L \), and mutates \( L \) by removing all instance of \( \text{target} \) in the list.

For example:

\[
\begin{align*}
A &= [] \\
B &= [1, 4, 13, 4, 15, 2, 6] \\
C &= ["Hello", "world", "I", "am", "not", "Ultron"]
\end{align*}
\]

\[
\begin{align*}
\text{clean_list}(A, "hello") &= 0 \\
A &= [] \\
\text{clean_list}(B, 4) &= 2 \\
B &= [1, 13, 15, 2, 6] \\
\text{clean_list}(C, "HAL 9000") &= 0 \\
C &= ["Hello", "world", "I", "am", "not", "Ultron"]
\end{align*}
\]
Use accumulative recursion to write the function `spread` that consumes a list of numbers, and returns the difference between the largest and smallest values in the list. For example:

```
spread([3, 1, 9, 17, -4, 2]) => 21
spread([2]) => 0
spread([]) => 0
```
Consider the number guessing game between two players, A and B. Player A thinks of a number between two values low and high, and player B tries to guess it. Player A will provide a hint telling B if their guess is too high or too low, which allows B to narrow the range for the secret number. Assume that B is smart, and will always guess halfway between the current range for low and high.

For example, suppose A chooses 80 as the secret value between 0 and 100. B initially guesses 50 = (0+100)/2, and A indicates the guess is too low, so B now knows that the secret number is in the range [51,100], and will then guess midway between 51 and 100 (e.g. round down to 75), and proceed on until 80 is identified.

Write a function `guesses_needed` that consumes integers `secret`, `low`, and `high` and determines how many guesses B needs to guess the secret number between `low` and `high`.

Assume that `0<=low<=high`.

How would you modify this to find the list of all the guesses, in order in which they would be made?