CS 5: **Putting loops to work...**


[ 26250, 5250, 1050, 210, ? ]

[ 90123241791111, 93551622, 121074, 3111, ? ]

[ 1, 11, 21, 1211, 111221, ? ]

**What's next?**

I'm glad you asked!

Homework 8: due Mon., 4/1 by midnight
Office hrs. Thurs! + lots of tutoring, LAC & ...
Midterm 4/2; review on the CS5 homepage

The **read it and weep** sequence

1
11
21
1211
111221
312211
31112221 ...

**str vs. int**

When does the first 4 appear?

How fast do these terms grow?

**Extra extra credit:** in wk10!

__hw8pr4: T. T. Securities (TTS)__

Analyzes a sequence of “stock prices”

L = [ 40, 80, 10, 30, 27, 52, 5, 15 ]

Implement a (text) menu:

(0) Input a new list
(1) Print the current list
(2) Find the average price
(3) Find the standard deviation
(4) Find the min and its day
(5) Find the max and its day
(6) Your TTS investment plan
(9) Quit

Enter your choice:

The **TTS advantage!**

Your stock’s prices: L = [ 40, 80, 10, 30, 27, 52, 5, 15 ]

<table>
<thead>
<tr>
<th>Day</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>40.0</td>
</tr>
<tr>
<td>1</td>
<td>80.0</td>
</tr>
<tr>
<td>2</td>
<td>10.0</td>
</tr>
<tr>
<td>3</td>
<td>30.0</td>
</tr>
<tr>
<td>4</td>
<td>27.0</td>
</tr>
<tr>
<td>5</td>
<td>52.0</td>
</tr>
<tr>
<td>6</td>
<td>5.0</td>
</tr>
<tr>
<td>7</td>
<td>15.0</td>
</tr>
</tbody>
</table>

Important fine print:

To make our business plan realistic, however, we only allow selling after buying.
Loops

Basic design strategies

Recursion

for two types

L = [3, 15, 17, 7]

for i in range(len(L)):
    L[i] = i

print L

for x in L:
    print x

Index-based loops

Element-based loops

def fac( N ):
    result = 1
    for x in range(1,N+1):
        result *= x
    return result

M.T. Review

See review problems @ the cs5 site

Picobot! There are NO Picobot questions on the exam...

Recursion!
• You should feel confident you could create recursive functions to solve small problems, e.g., scalable-scoring, finding sum or max or min, computing power or factorial (or variations)
• You should be comfortable manipulating lists and strings (indexing, slicing, slinging with "splitter"), What’s L[1:1-1]?
• You should understand how to use, read, and compile list comprehensions, e.g., LC = [ i**2 for i in L ]
• You should be able to design solutions using the "list of lists" technique (LOL), which uses list comprehensions, e.g., to find the highest scalable-scoring word among a list or the lowest scalable-scoring of a shifted string. Basic syntax: List actual[ i ] for x in L
• You should understand the use-it-or-lose-it design strategy, e.g., the LSC, exact change, jotto, and sort homework problems
• Turtle graphics will get be on the exam
• Understand the difference between print and return
• Look over how you compare larger programs out of smaller ones + how inputs and outputs are used, e.g., for Caesar Cipher

Representing Data!!!
• Remind yourself of the various types of data (int, str, float, etc.), how they’re different and how to convert between types
• Know how characters are represented with char and ord
• Be comfortable with base-2 arithmetic (and base 10), along with how to convert from base to base (big balanced ternary)
• Remind yourself of how the bits of a base-2 number influence its value – what do right- and left-shifting do?

You may use functions from class and homework on the exam without reimplementing them. Here are some, but not all, of them:

• removeOne( e, L ], removeAll( e, L ] , removeUptil( e, L ] , count( e, L ]
• ind( a, L ] , frontnum( L ], binToNum( binStr ] , numToBin( n ] , ...
**hw8pr4: T. T. Securities (TTS)**

Analyzes a sequence of "stock prices"

\[
\begin{array}{cccccccc}
  \text{day} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
  L = & [ & 40, & 80, & 10, & 30, & 27, & 52, & 5, & 15 ] & \\
\end{array}
\]

Implement a (text) menu:

(0) Input a new list
(1) Print the current list
(2) Find the average price
(3) Find the standard deviation
(4) Find the min and its day
(5) Find the max and its day
(6) Your TTS investment plan
(9) Quit

Enter your choice:

**User input...**

```python
meters = input('How many m? ')
cm = meters * 100
print("That's", cm, 'cm.\')
```

*What will Python think?*

**Fix #1: convert to the right type**

```python
m_str = input('How many m? ')
meters = float( m_str )
cm = meters * 100
print('That is', cm, 'cm.\')
```

Fix #2: **convert and check**

```python
m_str = input('How many m? ')

try:
meters = float( m_str )
except:
meters = 42.0
print("That's", cm, 'cm.\')
```

```
'42'
name: m_str type: string

42.0
name: meters type: float

4200.0
name: cm type: float

... but crash-able
```

try-except lets you try code and – if it crashes – catch an error and handle it
Fix #3: **eval** executes Python code!

```python
m_str = input('How many m? ')
meters = eval(m_str)

cm = meters * 100
print('That is', cm, 'cm.')
```

---

**A larger application**

```python
def menu():
    """ prints our menu of options """
    print('(0) Continue')
    print('(1) Enter a new list')
    print('(2) Analyze')
    print('(9) Break (quit)')

def main():
    """ handles user input for our menu """
    while True:
        menu()
        uc = input('Which option? ')
        try:
            uc = int(uc)
        except:
            continue
        if uc == 0:
            # was it an int?
            # back to the top!
            continue
        elif uc == 9:
            # Break (quit)
            break
        elif uc == 2:
            # Analyze!
            # and so on ...
            continue
        elif uc == 1:
            # Get new list
            # and so on ...
            continue
        elif uc == 3:
            # Unannounced menu option!
            print('Invalid option.')
            continue
        else:
            # Unannounced menu option (problem with your code)
            print('Invalid option.')
            continue
    print('Thanks for using Predictor!')
```

---

**Try it!**

Full program example of user-interactions

```python
Name(s) ___________
```

---

```
 Called a helper function

#3: What code below handles an input of 3? Why?

```
Functions you'll write

All use loops...

Menu

(0) Input a new list
(1) Print the current list
(2) Find the average price
(3) Find the standard deviation
(4) Find the min and its day
(5) Find the max and its day
(6) Your TTS investment plan
(9) Quit

Enter your choice:

```
def average(L):
    total = sum(L)
    average = total / len(L)
    return average

def stdev(L):
    L_av = average(L)
    variance = sum((L[i] - L_av)**2) / len(L)
    stdev = variance**0.5
    return stdev
```

```
def minday(L):
    min_price = L[0]
    min_day = 0
    for i in range(len(L)):
        if L[i] < min_price:
            min_price = L[i]
            min_day = i
    return min_price, min_day

def maxday(L):
    max_price = L[0]
    max_day = 0
    for i in range(len(L)):
        if L[i] > max_price:
            max_price = L[i]
            max_day = i
    return max_price, max_day
```

hw9pr4: T. T. Securities (TTS)

Analyzes a sequence of "stock prices"

```
L = [ 40, 80, 10, 30, 27, 52, 5, 15 ]
```

Min price

```
L = [ 40, 80, 10, 30, 27, 52, 5, 15 ]
```

What's the idea for finding the smallest (minimum) price?

track the value of the minimum so far as you loop over L
Min price vs. min day

\[ L = [40, 80, 10, 30, 27, 52, 5, 15] \]

**def minprice(L):**

\[
m = L[0] \\
for x in L: \\
    if x < m: \\
        m = x \\
return m
\]

What about tracking BOTH the day of the minimum price and that min price?

**Try it!**

**minday** = 0

\[ L = [40, 80, 10, 30, 27, 52, 5, 15] \]

**def minday(L):**

\[
\begin{align*}
\text{minday} &= 0 \\
\text{minval} &= 40 \\
&= 10 \\
&= 5
\end{align*}
\]

**minday(L):**

track both price and day

\[ \text{minval} = L[0] \]

\[ \text{minday} = 0 \]

\[ \text{for } i \text{ in range(len}(L)):\]

\[
\begin{align*}
\text{if } & \\
\text{update both } & \text{as needed)
\end{align*}
\]

**return minprc, minday**

**def min_prc_day(L):**

\[
\begin{align*}
\text{minprc} &= L[0] \\
\text{minday} &= 0 \\
\text{for } i \text{ in range(len}(L)):\]

**return both!**

**def mindiff(L):**

\[
\text{mdiff} = \text{abs}(L[1]-L[0]) \\
\text{for } i \text{ in range(len}(L)):\]

**Hint:** Use nested loops:

for i in range(4):
    for j in range(4):
        Track the value of the minimum so far as you loop over L twice...

**return mdiff**

**mindiff([42, 3, 100, -9, 7]):**

4

5 is returned

6 is returned

5 is returned

6 is returned
Investment analysis for the 21st century ... and beyond

T. T. Securities

(0) Input a new list
(1) Print the current list
(2) Find the average price
(3) Find the standard deviation
(4) Find the min and its day
(5) Find the max and its day
(6) Your TTS investment plan
(9) Quit
Enter your choice:

Software side ...

Hardware side...

The TTS advantage!

Your stock's prices:

<table>
<thead>
<tr>
<th>Day</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>40.0</td>
</tr>
<tr>
<td>1</td>
<td>80.0</td>
</tr>
<tr>
<td>2</td>
<td>10.0</td>
</tr>
<tr>
<td>3</td>
<td>30.0</td>
</tr>
<tr>
<td>4</td>
<td>27.0</td>
</tr>
<tr>
<td>5</td>
<td>52.0</td>
</tr>
<tr>
<td>6</td>
<td>5.0</td>
</tr>
<tr>
<td>7</td>
<td>15.0</td>
</tr>
</tbody>
</table>

To make our business plan realistic, however, we only allow selling after buying.

The TTS advantage!

Your stock's prices:

<table>
<thead>
<tr>
<th>Day</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>40.0</td>
</tr>
<tr>
<td>1</td>
<td>80.0</td>
</tr>
<tr>
<td>2</td>
<td>10.0</td>
</tr>
<tr>
<td>3</td>
<td>30.0</td>
</tr>
<tr>
<td>4</td>
<td>27.0</td>
</tr>
<tr>
<td>5</td>
<td>52.0</td>
</tr>
<tr>
<td>6</td>
<td>5.0</td>
</tr>
<tr>
<td>7</td>
<td>15.0</td>
</tr>
</tbody>
</table>

To make our business plan realistic, however, we only allow selling after buying.

Important fine print:

For each buy-day, b:
- set max-so-far = 0
- for each sell-day, s:
  - compute the profit
  - if profit is > max-so-far:
    - remember it in a variable!
- return profit, its b-day, and s-day

Important fine print:

For each sell-day, s:
- compute the profit
- if profit is > max-so-far:
  - remember it in a variable!
- return profit, its b-day, and s-day

CS5 wishes future you a thoroughly profit-filled week ...

... and hw8 is ready to help!
M.T. Review

Recursion!!

- You should feel confident you could create recursive functions to solve small problems, e.g., scrabble-scoring, finding sum or max or min, computing power or factorial (or variations)
- You should be comfortable manipulating lists and strings (indexing, slicing, slicing with a "stride") What's `s[1::1:2]`?
- You should understand how to use, read, and compose list comprehensions, e.g., `LC = [x**2 for x in L]`
- You should be able to design solutions using the "list of lists" technique (LoL), which uses list comprehensions, e.g., to find the highest scrabble-scoring word among a list or the lowest scrabble-scoring of a shifted string. Basic syntax: `LoL = { [sc(x),x] for x in L }
- You should understand the *use-it-or-lose-it* design strategy, e.g., the LCS, exact_change, jotto, and sort homework problems
- Turtle graphics will **not** be on the exam
- Understand the difference between `print` and `return`
- Look over how you composed larger programs out of smaller ones + how inputs and outputs are used, e.g., for Caesar Cipher

Representing Data!!! 😛

- Remind yourself of the various types of data (int, str, float, etc.), how they're different and how to convert between types
- Know how characters are represented with `chr` and `ord`
- Be comfortable with base-2 arithmetic (and base 10!), along with how to convert from base to base (no balanced ternary)
- Remind yourself of how the bits of a base-2 number influence its value – what do right- and left-shifting do?

Circuits and Assembly!!!!

- Know how the basic logic gates operate (AND and OR for any # of inputs, NOT for 1 input only)
- Know how minterm expansion works (an OR of ANDs, each selecting one input) and how it enables the engineering of any circuit at all, given its truth-table specification
- You should be comfortable going from a truth-table to a circuit (using minterm expansion) and going from a circuit to a truth-table (ditto)
- You should be able to write simple looping programs in Hmmm, e.g., factorial, power, Fibonacci, ...
- You should know what the stack is, what it's used for conceptually (holding functions' variables and data) You should know that `pushr` nor `popr` store and load data to the stack, but won't have to write code using them. More specifically, the exam will **not** ask you to implement recursion or stack-based functions in Hmmm. There would only be looping (jumping) + conditional examples, e.g., What is `jneqzn r2 42`?
- The exam will have a full Hmmm reference (you don't need your own)

Loops 

- You should feel comfortable with how for loops (both element-based and index-based) and while loops work – and be able to compose small functions that use them, e.g., ones similar to those in the pi-estimation and TT securities (statistics) problems.
- You should understand how nested loops work and be able to read or compose examples (such as the TTS strategy or our in-class problems)

You may use functions from class and hw on the exam without reimplementing them. Here are some, but not all, of them:

- `removeOne(e, L)`
- `removeAll(e, L)`
- `removeUpto(e, L)`
- `count(e, L)`
- `ind(e, L)`
- `frontNum(L)`
- `binToNum(binstr)`
- `numToBin(n)` ...
(A) Which code below handles an input of 5? of 7?

# example looping program

```python
def menu():
    """a function that simply prints the menu """
    print()
    print("(0) Continue!")
    print("(1) Enter a new list")
    print("(2) Predict the next element")
    print("(9) Break! (quit)")
    print()

def main():
    """the main user loop """
    print()
    print("+++++++++++++++++++++++++"
    print("Welcome to the PREDICTOR!")
    print("+++++++++++++++++++++++++")
    print()

secret_value = 4.2
L = [30,10,20] # an initial list

while True:
    # the user-interaction loop
    print("\n\nThe list is", L)
    menu()
    uc = input( "Choose an option: " )

    # "clean and check" the user's input
    #
    try:
        uc = int(uc) # make into an int!
    except:
        print("I didn't understand your input! Continuing...")
        continue

    # run the appropriate menu option
    #
    if uc == 9:
        # we want to quit
        break
    elif uc == 0:
        # we want to continue...
        continue
    elif uc == 1:
        # we want to enter a new list
        newL = eval(newL) # eval runs Python's interpreter! Note: Danger if type(newL) != type([]):
        print("That didn't seem like a list. Not changing L.")
        L = newL # here, things were OK, so let's set our list, L
    elif uc == 2:
        # predict and add the next element
        n = predict(L) # get the next element from the predict function
        print("The next element is", n)
        L = L + [n] # and add it to the list
    elif uc == 3:
        # unannounced menu option!
        pass # this is the "nop" (do-nothing) statement in Python
    elif uc == 4:
        # unannounced menu option (slightly more interesting...)
        m = find_min_loc(L)
        print("The minimum value in L is", m)
    elif uc == 5:
        # another unannounced menu option (even more interesting...)
        minval, minloc = find_min_loc(L)
        print("The minimum value in L is", minval, "at day ", #, minloc)
    elif uc != 1 and uc != 2 and uc != 3 and uc != 4 and uc != 0 and uc != 9:
        # if the input uc was anything else
        print(uc, "? That's not on the menu!"
    print("Running again..."

print("\nI predict... \n\n... that you'll be back!
")
```

(B) What does choice 3 print that 0 does not?

```
def menu():
    """a function that simply prints the menu """
    print()
    print("(0) Continue!")
    print("(1) Enter a new list")
    print("(2) Predict the next element")
    print("(9) Break! (quit)")
    print()
```

(C) What line of code runs after this break?

```
if uc == 9:
    # we want to quit
    break
```

(D) What could you input for newL that would print this?

```
elif uc == 1: # we want to enter a new list
    newL = input("Enter a new list: ") # enter _something_
```

(E) What could you type for newL that would print this?

```
elif uc == 1: # we want to enter a new list
    newL = input("Enter a new list: ") # enter _something_
```

(EC) How could a user learn the value of secret_value if they knew that variable name and could run the program -- but didn't have this source code?

```
secret_value = 4.2
```

(Full program example of user-interactions)