Welcome back!

My spring break: Meet Peaches (and her 7 babies!)
CS 5: *Putting loops to work...*

\[
\begin{bmatrix}
-35, & -24, & -13, & -2, & 9, & 20, & 31, & ? \\
26250, & 5250, & 1050, & 210, & ? \\
90123241791111, & 93551622, & 121074, & 3111, & ? \\
1, & 11, & 21, & 1211, & 111221, & ? \\
\end{bmatrix}
\]

*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
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., 11/5 by midnight
Office hrs. Fri! + lots of tutoring, LAC &...
Midterm 11/8; review on the CS5 homepage
The **read it and weep** sequence

1
11
21
1211
111221
312211
13112221
13213211
...

**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"

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:

url
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

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

Recursion

```
def fac( N ):
    if N == 1:
        return 1
    else:
        return N*fac(N-1)
```

Is one more reasonable than the other?
Loops

Design strategy: look for repetition + describe it...

Recursion

Is one more reasonable than the other?

Design strategy: look for self-similarity + describe it...

def fac( N ):
    if N == 1:
        return 1
    else:
        return N*fac(N-1)
for: two types

\[ L = [3, 15, 17, 7] \]

for \( x \) in \( L \):
    print \( x \)

"deceptively easy"

element-based loops
for: two types

\[ L = [3, 15, 17, 7] \]

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

for \( x \) in \( L \):
    print \( x \)
for: two types

\[ L = [3, 15, 17, 7] \]

\[
\begin{align*}
\text{for } x \text{ in } L: \\
\text{print } x
\end{align*}
\]

element-based loops

\[
\begin{align*}
\text{for } i \text{ in } \text{range(len}(L)) \\
\text{print } L[i]
\end{align*}
\]

index-based loops

Elements vs Indexes

Indices
for: two types

L = [3, 15, 17, 7]

for x in L:
    print x

element-based loops

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

index-based loops

"Get into a rut... and stay there!"

Elements vs Indexes

Indices

"Get into a rut... and stay there!"
**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!!!** 🍎:i<3 three-eyed punctuation:
- 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?

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

```python
removeOne( e, L ), removeAll( e, L ), removeUpto( e, L ), count( e, L ),
ind( e, L ), frontNum( L ), binToNum( binstr ), numToBin( n ), ...
```

**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 (your 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)

(*) For up to **two points!** of extra credit on the midterm… Pair up with s/o and find something you have in common (that you didn't know before!)

**Names:** __________________________

**Thing in common:** __________________________

Then, read these over & look for ones **least** familiar...
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., 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., finding the highest scrabble-scoring word among a list or the lowest scrabble-scoring of a shifted string.
• You should understand how characters are represented with chr and ord
• 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.

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 a truth-table (ditto)
• You should be comfortable going from a circuit to a truth-table (ditto) and going from a truth-table to a circuit (using minterm expansion) conceptually
• You should be able to write simple looping programs in Hmmm, 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 ), ...
hw8pr4: T. T. Securities (TTS)

Analyzes a sequence of "stock prices"

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:

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

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

```
meters = int(input('How many m? '))
```

```
cm = int(meters) * 100
```

```
print("That's", cm, 'cm.')
```

```
That's 500 cm.
That's 5555555...5 cm.
```

What will Python think?

I think I like these units better than light years per year!
meters = input('How many m? ')
cm = meters * 100
print('That is', cm, 'cm!')

I think I like these units better than light years per year!

What will Python think?
Fix #1: convert to the right type

```python
m_str = input('How many m? ') # CS 5!

meters = float(m_str)

cm = meters * 100

print('That is', cm, 'cm."

... but crash-able
```
Fix #2: **convert and check**

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

try:
    meters = float(m_str)
except:
    print("What? Didn't compute!")
    print("Setting meters = 42")
meters = 42.0

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

**try-except** lets you try code and – if it crashes – catch an error and handle it.
Fix #2:

These errors are called **exceptions**. This is **exception handling**.

```python
try:
    meters = float(m_str)
except:
    print("What? Didn't compute!")
    print("Setting meters = 42")
    meters = 42.0

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

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

```python
>>> Six
! Unknown variable!

m_str = input('How many m? ')

meters = eval( m_str )

cm = meters * 100

print('That is', cm, 'cm.')

```

eval('5') → 5
eval('5+5') → 10

eval('Six')

Six = 7
eval('Six')

→ 'Six'
```

What could go wrong here?
Fix #3: `eval` executes Python code!

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

try:
    meters = eval(m_str)
except:
    print("What? Didn't compute!"")
    print("Setting meters = 42")
meters = 42.0

cm = meters * 100
print('That is', cm, 'cm.')
```
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)  # was it an int?
        except:
            continue  # back to the top!
def main():
    """ handles user input for our menu """
L = [30,10,20]  # a starting list

    while True:
        menu()  # print menu
        uc = input('Which option? ') ...

        if uc == 9:
            (9) Quit

        elif uc == 0:
            (0) Continue

        elif uc == 1:
            (1) Get new list

        elif uc == 2:
            (2) Analyze !  ... and so on ...
def main():
    """ handles user input for our menu """
    L = [30,10,20]  # a starting list

    while True:
        menu()  # print menu
        uc = input('Which option? ')

        if uc == 9:
            break
        elif uc == 0:
            continue
        elif uc == 1:
            ... input ... eval ...
        elif uc == 2:

(9) Quit
(0) Continue
(1) Get new list
(2) Analyze !

break jumps out of the loop
continue jumps back to the top
uses eval (+check) for a new L
other functions as needed...
... and so on ...
**Try it!**

### Full program example of user-interactive loops

```python
# example looping program

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("(3) Find minimum")
    print("(4) Find maximum")
    print("(5) Exit")
    print()

def main():
    """the main user interaction loop """
    print("Welcome to the PREDICTOR!")
    print("(2) Predict the next element")
    print("(0) Continue!")
    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 == 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)
        else:
            # if the input uc was anything else
            print(uc, "? That's not on the menu!"")
            continue

        # print(uc, "?"")
        # # any other code here
        
        print("Running again\n")
        print("\nI predict... \n\n... that you'll be back!")

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

### Name(s)

```
input
```

### (A) Which code below handles an input of 5? of 7?

1. `if uc == 5: # another unannounced menu option (even more interesting...)`
2. `elif uc == 5: # another unannounced menu option (even more interesting...)`
3. `elif uc == 5: # another unannounced menu option (even more interesting...)`
4. `elif uc == 5: # another unannounced menu option (even more interesting...)`
5. `elif uc == 5: # another unannounced menu option (even more interesting...)`

**Answer:** 2

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

1. `print("I didn't understand your input. Continuing...")`
2. `print("I didn't understand your input. Continuing...")`
3. `print("I didn't understand your input. Continuing...")`
4. `print("I didn't understand your input. Continuing...")`
5. `print("I didn't understand your input. Continuing...")`

**Answer:** 1

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

1. `elif uc == 9: # we want to quit
   break # leaves the while loop altogether`
2. `elif uc == 9: # we want to quit
   break # leaves the while loop altogether`
3. `elif uc == 9: # we want to quit
   break # leaves the while loop altogether`
4. `elif uc == 9: # we want to quit
   break # leaves the while loop altogether`
5. `elif uc == 9: # we want to quit
   break # leaves the while loop altogether`

**Answer:** 2

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

1. `newL = eval(newL) # eval runs Python's interpreter! Note: Danger`  
2. `newL = eval(newL) # eval runs Python's interpreter! Note: Danger`  
3. `newL = eval(newL) # eval runs Python's interpreter! Note: Danger`  
4. `newL = eval(newL) # eval runs Python's interpreter! Note: Danger`  
5. `newL = eval(newL) # eval runs Python's interpreter! Note: Danger`  

**Answer:** 1

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

1. `secret_value`
2. `secret_value`
3. `secret_value`
4. `secret_value`
5. `secret_value`

**Answer:** 1

### (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?

1. By inspecting the code and finding the line where `secret_value` is assigned.
2. By using Python's `dir()` function to list all available variables.
3. By using Python's `help()` function on the `main` function.
4. By examining the program's output for any mention of `secret_value`.
5. By manually searching the code for any instance of `secret_value`.  

**Answer:** 2
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:

```python
def average(L):
    sum(L)
    len(L)
    \sum_{i=0}^{\text{len}(L)} (L[i] - L_{\text{av}})^2
    \sqrt{1/n \sum_{i=0}^{\text{len}(L)} (L[i] - L_{\text{av}})^2}

def stdev(L):
    \sqrt{1/n \sum_{i=0}^{\text{len}(L)} (L[i] - L_{\text{av}})^2}

def minday(L):

def maxday(L):
```

webbrowser.open_new_tab(url)
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:
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 )
```

```
def stdev( L )
```

```
def minday( L )
```

```
def maxday( L )
```

\[
\sqrt{\frac{\sum_{i} (L[i] - L_{av})^2}{len(L)}}
\]
Min price

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

\[ m = L [0] \]

\( m \) is the "min so far"

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 ]
```

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

```
def minprice( L ):
    m = L[0]
    for x in L:
        if x < m:
            m = x
    return m
```
def min_prc_day( L ):
    minprc = L[0]
    minday = 0
    for i in range(len(L)):
        if
            return minprc, minday

def mindiff( L ):
    mdiff = abs(L[1]-L[0])
    for
        for
            if
                return mdiff

Finish this code to return both the minprice and the minday of $L$!

Expand on the minprice example...

Try it!

Write mindiff to return the smallest absolute difference between any two elements from $L$.

Only consider abs differences. $L$ will be a list of 2 or more #s. Hint: Use a nested loop!
The function `min_prc_day(L)` finds the minimum price and its corresponding day in a list of prices:

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

```python
def min_prc_day(L):
    minprc = L[0]
    minday = 0
    for i in range(len(L)):
        if some_condition:
            # Update both price and day
    return minprc, minday
```

The function tracks both the minimum price and the corresponding day, updates them as needed, and returns both values.

6 is returned

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

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

6 is returned

5 is returned
Write `mindiff` to return the **smallest** abs. diff. between any two elements from `L`.

```python
def mindiff( L):
    mdiff = abs(L[1]-L[0])
    for i in range(len(L)):
        for j in range(i+1,len(L)):
            if
    return mdiff
```

**Example:**

```python
mindiff( [42,3,100,-9,7] )
```

4

\[
\begin{array}{cc}
1 & 4 \\
\uparrow & \uparrow \\
\downarrow \downarrow & \downarrow \downarrow \\
\end{array}
\]

**Hint:** Use nested loops:

```python
for i in range(4):
    for j in range(4):
        if
```

Track the value of the **minimum so far** as you loop over \( L \) **twice**...
Write `mindiff` to return the **smallest** abs. diff. between any two elements from `L`.

```python
def mindiff( L ):
    mdiff = abs(L[1]-L[0])
    for i in range(len(L)):
        for j in range(i+1,len(L)):
            if abs(L[j]-L[i]) < mdiff:
                mdiff = abs(L[j]-L[i])
    return mdiff
```

**Hint:** Use nested loops:
```python
for i in range(4):
    for j in range(4):
```

Track the value of the _minimum so far_ as you loop over `L` _twice_...
T. T. Securities

"Taking the broke out of brokerage."

Software side ...

Investment analysis for the 21st century ... and beyond

(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.
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>

set max-so-far = 0

for each buy-day, \( b \):

for each sell-day, \( s \):

compute the profit

if profit is > max-so-far:

\[ \text{remember it in a variable!} \]

return profit, its b-day, and s-day

Important fine print:

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>

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

---

**Important fine print:**

For each buy-
day, 

compute the 
profit 
if 
profit 
is > max-
so-far: 

remember it 
in a variable!

return profit, its b-day, and s-day

---

... and hw8 is ready to help!

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