Welcome back!

My spring break: Meet Peaches (and her 7 babies!)
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
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<

111312213211<

...```

**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

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

Basic design strategies

Recursion

```python
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

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

---

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

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

$L = [3, 15, 17, 7]$
**for:** two types

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

for \( i \) in range(len(L))

print L[i]

for \( x \) in L:

print \( x \)
for: two types

L = [3, 15, 17, 7]

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

index-based loops

for \(x\) in L:
print \(x\)
element-based loops

Elements vs Indexes

Indices
for: two types

L = [3, 15, 17, 7]

for x in L:
    print x

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

Elements vs Indexes

"Get into a rut... and stay there!"
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., 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
- Look over how you composed larger programs out of smaller ones + how inputs and outputs are used, e.g., for Caesar Cipher

Representing Data!!! 🍓 1<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?

(∗) 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...

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 ), ...
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., for finding the highest scrabble-scoring word among a list or the lowest scrabble-scoring of a shifted string.
- 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., for finding the highest scrabble-scoring word among a list or the lowest scrabble-scoring of a shifted string.
- 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), 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 minterms work (an OR of ANDs, each selecting one input) and how they're used to create any circuit at all, given its truth-table (ditto) 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 is available conceptually or in Hmmm, 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 ), ...
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:
meters = int(input('How many m? '))
meters = int(meters)

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

That's 500 cm.
That's 555555555...5 cm.

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

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

input ALWAYS returns a string – no matter what's typed!
```

What will Python think?

I think I like these units better than light years per year!
Fix #1: **convert** to the right type

\[
m\_str = \text{input}(\text{'How many m? '} )
\]

\[
meters = \text{float}( m\_str )
\]

\[
cm = meters * 100
\]

\[
\text{print}(\text{'That is'}, cm, \text{', 'cm.'})
\]
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:
    m_str = input('How many m? ')
    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
m_str = input('How many m? ')
meters = eval(m_str)

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

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

\[m_{\text{str}} = \text{input}('\text{How many m? '}\)\]

```python
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.')
```

**What could REALLY go wrong here?**

A lot!!!
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)  # was it an int?
        except:
            continue  # back to the top!
```

Perhaps `uc` the reason for this?
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 ...
            break

        elif uc == 2:
            ... and so on ...

    (9) Quit
    (0) Continue
    (1) Get new list
    (2) Analyze !
def main():
    """ 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()

    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 # leaves the while loop altogether
    elif uc == 0: # we want to continue...
        continue # goes back to the top of the while loop
    elif uc != 3: # unannounced menu option!
        pass
    else:
        L = newL # here, things were OK, so let's set our list, L

    # "clean and check" the user's input
    #
    try:
        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.")
        else:
            L = newL # here, things were OK, so let's set our list, L
    except:
        print("I didn't understand your input. Not changing L.")

    # 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

    # 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!")

    print("Running again\n")

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

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

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

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

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

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

(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?
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:

\[ \text{def average( } L \text{ )} \]
\[ \text{def stdev( } L \text{ )} \]
\[ \text{def minday( } L \text{ )} \]
\[ \text{def maxday( } L \text{ )} \]

\[ \sqrt{\frac{\sum_{i=0}^{n} (L[i] - \text{average})^2}{\text{len}(L)}} \]
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:

```python
def average(L):
    return sum(L) / len(L)

def stdev(L):
    avg = average(L)
    variance = sum((x - avg)**2 for x in L) / len(L)
    return (variance)**0.5

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

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

Min price

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

track the value of the \textit{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?
Finish this code to return **both** the minprice and the minday of L!

Expand on the minprice example...

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

min_prc_day( [9, 8, 5, 7, 42] )
5, 2

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

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

mindiff( [42,3,100,-9,7] )
4

Try it!

Only consider abs differences. L will be a list of 2 or more #s.

**Hint:** Use a nested loop!
def min_prc_day(L):
    minprc = L[0]
    minday = 0
    for i in range(len(L)):
        if
            return minprc, minday
    return both

6 is returned

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

minday = 0 -> 2 -> 6
minval = 40 -> 10 -> 5

track both price and day
loop over
update both (as needed)
return both!
```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
```

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

5 is returned

```
6 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(len(L)):
            if
    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**...
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:
```
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 ...

Hardware 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>

**Important fine print:**

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

---

**set max-so-far = 0**

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

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

compute the \textit{profit}  

if \textit{profit} is \( > \) max-so-far:  

\textit{remember it in a variable!}  

return \textit{profit}, its \textit{b}-day, and \textit{s}-day
The TTS advantage!

Your stock's prices: \( L = [40, 80, 10, 30, 27, 52, 5, 15] \)

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

Important fine print:

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