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

[-35, -24, -13, -2, 9, 20, 31, ?]

[26250, 5250, 1050, 210, ?]

[90123241791111, 93551622, 121074, 3111, ?]

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

What's next?

I'm glad you asked!

Reading: Section 5.5

Loops

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

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

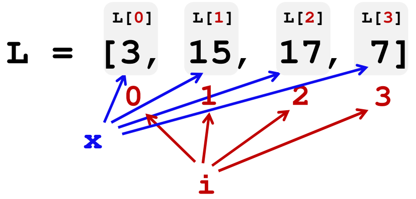
Is one more *reasonable* than the other?

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

Recursion

```
def fac(N):
    if N == 0:
        return 1
    else:
        return N * fac(N - 1)
```

elements vs. indices



```
def sum(L):
  total = 0
  for x in L:
    total += x
  return total
```

element-based loops

```
def sum(L):
  total = 0
  for i in range(len(L)):
    total += L[i]
  return total
```

list

index-based loops

User input...

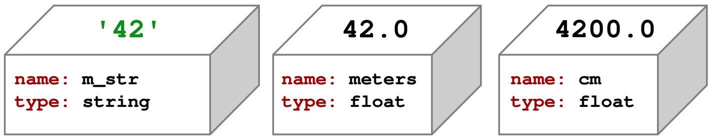
```
meters = input('How many m? ')

cm = meters * 100

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

What will Python think?

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

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.')
```



... but crash-able

Fix #2: convert and check

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

try:
  meters = float(m_str)
except:
  print("What? Does not compute!")
  print("I don't get", m_str)
  print("Setting meters = 4
  meters = 42.0

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

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? ')

try:
  meters = eval(m_str)
except:
  print("What? Does not compute!")
  print("I don't get", m_str)
  print("Setting meters = 42")
  meters = 42.0

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

What could go wrong here?

A larger application

```python
def menu():
    """Prints our menu of options."""
    print("(0) Continue")
    print("(1) Enter a new list")
    print("(2) Predict")
    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:
            print("I didn't understand that")
            continue           # Back to the top!
```

Calls a helper function

Perhaps uc the reason for this?

Functions you'll write

All use loops...

Menu

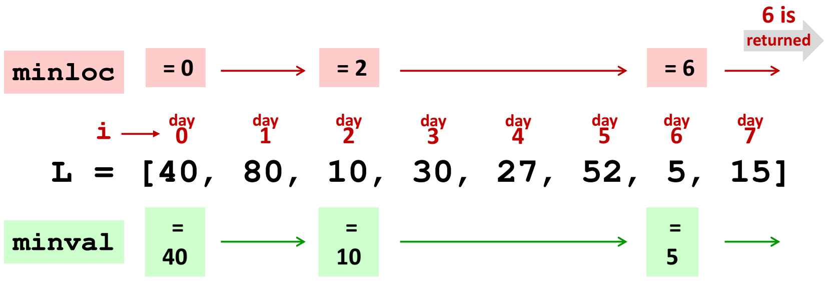
```
(0) Input a new list
(1) Print the current list
(2) Find the average price
(3) Find the standard deviation
(4) Find the min and its day
(5) Find the max and its day
(6) Your TTS investment plan
(9) Quit
Enter your choice:
```

```
def average(L)

def stdev(L)
```

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

```
def minday(L)

def maxday(L)
```

```
import webbrowser
webbrowser.open_new_tab(url)
```



```
def i_min(L):
  minval = L[0]
  minloc = 0
  for i in range(len(L)):
    if

  return minloc
```

track *both* day and price

loop!

update when needed

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

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

4

L

```
def mindiff(L):

  m = abs(L[1]-L[0])

  for i in range(len(L)):
    for j in range(   , len(L)):

      if

  return m
```

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**...

The TTS advantage!

What is the best TTS investment strategy here?

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

Day	Price
0	40.0
1	80.0
2	10.0
3	30.0
4	27.0
5	52.0
6	5.0
7	15.0

```
for each buy-day, b:
  for each sell-day, s:
    compute the profit
    if it's the max-so-far:
      remember it in a variable!
```

Important fine print:

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

Full program example of user interactions

```
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():
    """A sample main user-interaction 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("\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
```

main function

while True:

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

(1) Which block below handles an input of 7 ?

(2) What does choice 0 *not* print that 3 *does*?

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

(5) 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_

            # "clean and check" the user's input
            #
            try:
                newL = eval(newL)    # Note: Danger!
                if type(newL) != type([]):
                    print("That didn't seem like a list. Not changing
L.")

                else:
                    L = newL  # things were OK, so let's set L
            except:
                print("I didn't understand your input. Not changing
L.")

        elif uc == 2:        # predict and add the next element
            n = predict(L)   # get next element from predict function
            print("The next element is", n)
            print("Adding it to your list...")
            L = L + [n]      # and add it to the list

        elif uc == 3:  # unannounced menu option!
            pass       # this is Python’s "nop" (do-nothing) statement

        elif uc == 4:  # intersting unannounced menu option
            m = find_min(L)
            print("The minimum value in L is", m)
        elif uc == 5:  # more interesting unannounced option
            minval, minloc = find_min_loc(L)
            print("The minimum in L is", minval, "at day #", minloc)
        else:
            print(uc, " ?      That's not on the menu!")

        # last line of while True loop
        print("\nRunning again... !\n")

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

(6) `predict` is a function defined elsewhere (off this page). Find the two other functions called here, but defined elsewhere. They both include *`find`* in their names!

(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 code*?

Finish this code to return the **index** (location) of L's min.

$$\underbrace{[9,\ 8,\ 5,\ 7,\ 42]}_{}{}^{L}$$

```
>>> i_min( [9, 8, 5, 7, 42] )
            0  1  2  3   4
2
```

```
def i_min(L):

  minval = L[0]
  minloc = 0

               list
  for i in range(len(L)):
     if ______________ :
        minval = _____
        minloc = _____

  return minloc
```

Hints:
track of the minimum value in minval
track the location of the min inside minloc

What does this print?

```
          list
for i in range(4):
              list
   for j in range(4):
      print(abs(i-j), end='')
   print()
```

j

0	1	2	3	
				0
				1
				2
				3

i

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

Only consider **abs** differences.
L will be a list of numbers.
Hint: Use a nested loop!

```
>>> mindiff( [42, 3, 47, 100, -9] ) ⟶ 5

def mindiff(L):
```

Quiz, p.2