CS 100: Python Lists and Function Return Values

Chris Kauffman

Week 6-1
Logistics

Reading

- Pattern Ch 5: Algorithms And Heuristics
- Think Ch 11: Lists (link)

Homework 4
Due next week

Mini-Exam 2 Today

Goals Today

- Python lists
- Returning things from Functions
Questions on Computability

- Is how can one determine whether a computer program finishes?
- Can one program determine if another computer program will terminate?
- How does a human stop a program from running?
Exercise: Review of Lists

Write some python code which will accomplish the following

- Create a list named `the_nums` with the numbers 2, 4, 8, 16
- Create a list named `the_names` with the strings Frank, Claire, and Doug in it
- Change the number at index 2 of `the_nums` to be 32
- Print only the number at index 1 of `the_names`
- Print both lists to the screen
- Print the length of both lists
- Loop through the list `the_nums` and print each item in it
Exercise: Average of Numbers

- Adapt the code for `max_number(L)` to find the average of the numbers in a list
- Call your function `list_average(L)`
- **Remember**: Exercise answers are usually distributed with the lecture slides
- Follow the *pattern* demonstrated in `max_number(L)` but will need to change some details

```python
# Find the maximum number and print it
def max_number(L):
    max = -1
    for number in L:
        if number > max:
            max = number
    print("The max is "+str(max))
```
def list_average(L): # Print the average of a list
    total = 0
    for num in L:
        total = total + num
    avg = total / len(L)
    print("Average is "+str(avg))

Problem: Printing doesn’t cut it
Suppose we want to compare the average scores of two classes in code?

scores_sec1 = [13,20,35,32,40]
scores_sec2 = [40,25,37,13,21,23,18]

if ?? :
    print("Sec 1 has a better average")
else:
    print("Sec 2 has a better average")
Solution: Don’t print, return answer

Within a function, the return statement allows an answer to be given back to whoever executed the function.

```python
def list_average(L):
    # Compute and return average of list
    total = 0
    for num in L:
        total = total + num
    avg = total / len(L)
    return avg  # return an answer: the average
    # print("Average is "+str(avg))  # Don’t print

scores_sec1 = [13,20,35,32,40]
scores_sec2 = [40,25,37,13,21,23,18]

avg_sec1 = list_average(scores_sec1)  # store the average of sec1
avg_sec2 = list_average(scores_sec2)  # store the average of sec2

if ?? :
    print("Sec 1 has a better average")
else:
    print("Sec 2 has a better average")
```

```
Drawing vs "Normal" Functions

**Drawing Functions**

- Mostly put things on the screen
- Almost never return stuff

```python
draw_house(100,"red","blue")
pen_up()
forward(200)
pen_down()
draw_house(200,"green","yellow")
```

**"Normal" Functions**

- Mostly don’t put stuff on the screen
  - No printing
  - No moving turtles
- Frequently return an answer

```python
avg1 = list_average(scores1)
avg2 = list_average(scores2)
report_averages(avg1,avg2)  # prints
all_scores = merge_lists(scores1,scores2)
max_score = max_number(all_scores)
report_max(max_score)  # prints
```
As programs get more complex, seeing how they work gets more difficult: more state is hidden. The Python Visualizer is a useful web site to help.

List Average on Visualizer: https://goo.gl/9MW54s
# Find the maximum number and print it

def max_number(L):
    max = -1
    for number in L:
        if number > max:
            max = number
    print("The max is "+str(max))
def exponentiate(base, exponent):

    ▶ Raise base to a given power
    ▶ Involves a loop and repeated multiplication
    ▶ Assume both numbers are integers (no fractions)
    ▶ Raising numbers to the zeroth power always gives 1

Examples

twoToFour = exponentiate(2, 4)  # 16
threeToFive = exponentiate(3, 5) # 243
eightToZero = exponentiate(8, 0) # 1
nineToThird = exponentiate(9, 3) # 729
Solution: Exponentiate

```python
# A function to raise base to the exponent power
def exponentiate(base, exponent):
    ans = 1
    for i in range(exponent):
        ans = ans * base
    return ans
```
Example: Binary to Decimal Conversion

Recall Conversion of binary numbers

\[110110_2 = 0 \times 1 + 1 \times 2 + 1 \times 4 + 0 \times 8 + 1 \times 16 + 1 \times 32\]
\[= 54\]

Python lists with 1’s / 0’s

\[\text{bin1} = [1,1,0,1,1,0]\]

# Convert binary list to decimal number
# def bin_to_dec(binaryL):
  ???

\[\text{dec1} = \text{bin_to_dec(\text{bin1})}\]
\[\text{print(\text{dec1})} \quad \# 54\]
Strategies

**Strategy: Front to Back**

bin1 = [1,1,0,1,1,0]

\[2^5 + 2^4 + 2^2 + 2^1\]

▶ Go from front to back
▶ range(len(BinaryL))
▶ Power decreases by 1 each iteration

**Strategy: Back to Front**

bin1 = [1,1,0,1,1,0]

\[2^1 + 2^2 + 2^4 + 2^5\]

▶ Go from back to front
▶ range(len(binaryL)-1,-1,-1)
▶ Power increase by 1 each iteration
Implementations

Strategy: Front to Back

def binary_to_decimal_backwards(binaryL):
    sum = 0
    pow = 0
    for i in range(len(binaryL)-1,-1,-1):
        if binaryL[i]==1:
            sum = sum + exponentiate(2,pow)
        pow = pow+1
    return sum

Strategy: Back to Front

def binary_to_decimal_forwards(binaryL):
    sum = 0
    pow = len(binaryL)
    for i in range(len(binaryL)):
        pow = pow-1
        if binaryL[i]==1:
            sum = sum + exponentiate(2,pow)
    return sum
Creating New Lists

Create a new empty list and fill it up with numbers

```python
my_list = []
for i in range(10):
    my_list.append(i)
print(my_list)

for i in range(10,-1,-1):
    my_list.append(i)
print(my_list)
```

Lists can `append(x)` things to their end
Exercise: Create a Reversed List

```python
def reverse_list(L):
    # Create a reversed copy of L
    rev = []  # Start with an empty list
    for i in range(len(L) - 1, -1, -1):  # Use a for loop from back to front of L
        rev.append(L[i])  # Append each element of L to the reversed list
    return rev  # Return the reversed list
```

**Examples**

```python
for1 = [1,2,3,4]
rev1 = list_reverse(for1)
# [4,3,2,1]

for2 = [1,1,0,1,1,0]
rev2 = list_reverse(for2)
# [0,1,1,0,1,1]
```
# Create and return a reversed list with the append method of lists

def list_reverse(L):
    rev = []
    for i in range(len(L)-1,-1,-1):
        rev.append(L[i])
    return rev
Exercise: Converting from Decimal to Binary

Recall the process to convert a decimal number to a binary number

\[
\begin{align*}
54 \div 2 &= 27 \quad \text{rem } 0 \\
27 \div 2 &= 13 \quad \text{rem } 1 \\
13 \div 2 &= 6 \quad \text{rem } 1 \\
6 \div 2 &= 3 \quad \text{rem } 0 \\
3 \div 2 &= 1 \quad \text{rem } 1 \\
1 \div 2 &= 0 \quad \text{rem } 1 \\
\end{align*}
\]

\[54_{10} = 110110_2\]

def dec_to_bin(decimal):
    # Convert the decimal number to a binary list
    # Use repeated integer division:
    # \text{quot} = \text{num} \div \text{divis}
    # And repeated remainder:
    # \text{rem} = \text{num} \mod \text{divis}
    # Append remainder to a list
    # Reverse list and return

def dec_to_bin(decimal):
    # Convert the decimal number to a binary list
    # Use repeated integer division:
    # \text{quot} = \text{num} \div \text{divis}
    # And repeated remainder:
    # \text{rem} = \text{num} \mod \text{divis}
    # Append remainder to a list
    # Reverse list and return

def dec_to_bin(decimal):
    # Convert the decimal number to a binary list
    # Use repeated integer division:
    # \text{quot} = \text{num} \div \text{divis}
    # And repeated remainder:
    # \text{rem} = \text{num} \mod \text{divis}
    # Append remainder to a list
    # Reverse list and return

def dec_to_bin(decimal):
    # Convert the decimal number to a binary list
    # Use repeated integer division:
    # \text{quot} = \text{num} \div \text{divis}
    # And repeated remainder:
    # \text{rem} = \text{num} \mod \text{divis}
    # Append remainder to a list
    # Reverse list and return

def dec_to_bin(decimal):
    # Convert the decimal number to a binary list
    # Use repeated integer division:
    # \text{quot} = \text{num} \div \text{divis}
    # And repeated remainder:
    # \text{rem} = \text{num} \mod \text{divis}
    # Append remainder to a list
    # Reverse list and return

def dec_to_bin(decimal):
    # Convert the decimal number to a binary list
    # Use repeated integer division:
    # \text{quot} = \text{num} \div \text{divis}
    # And repeated remainder:
    # \text{rem} = \text{num} \mod \text{divis}
    # Append remainder to a list
    # Reverse list and return

def dec_to_bin(decimal):
    # Convert the decimal number to a binary list
    # Use repeated integer division:
    # \text{quot} = \text{num} \div \text{divis}
    # And repeated remainder:
    # \text{rem} = \text{num} \mod \text{divis}
    # Append remainder to a list
    # Reverse list and return

def dec_to_bin(decimal):
    # Convert the decimal number to a binary list
    # Use repeated integer division:
    # \text{quot} = \text{num} \div \text{divis}
    # And repeated remainder:
    # \text{rem} = \text{num} \mod \text{divis}
    # Append remainder to a list
    # Reverse list and return

def dec_to_bin(decimal):
    # Convert the decimal number to a binary list
    # Use repeated integer division:
    # \text{quot} = \text{num} \div \text{divis}
    # And repeated remainder:
    # \text{rem} = \text{num} \mod \text{divis}
    # Append remainder to a list
    # Reverse list and return

dec1 = 54
bin1 = dec_to_bin(dec1)
# [1, 1, 0, 1, 1, 0]
dec2 = 87
bin2 = dec_to_bin(87)
# [1, 0, 1, 0, 1, 1, 1]
# Convert a decimal number to a binary list

def dec_to_bin(decimal):
    digits = []
    while decimal > 0:
        remainder = decimal % 2
        decimal = decimal // 2
        digits.append(remainder)
    digits_rev = list_reverse(digits)
    return digits_rev
HW 4

- Only 3 problems
- Problems 1 and 2: Write a word-list processing functions
- Problem 3: Use code I provide and your functions to rank web pages, compare to Google search results
- May want to do some research on how web search engines rank web pages
- Zyante: Section 5.7 has some information, may want to look elsewhere also for info
- More discussion on Internet and Search later in the class
def count_odds(alist):
    ???

    how_many_odds = count_odds([1,2])
    print(how_many_odds) # 1

    how_many_odds = count_odds([8,6,7,5,3,0,9])
    print(how_many_odds) # 4

Sub-problems: How to...

- Examine each element in a list?
- Check if a number is odd?
- Update a total?
- Return an answer from a function?
HW Relevant Exercise: Find all Odd Numbers

def get_all_odds(num_list):
    ??

print( get_all_odds([2,4,6]) ) # []
print( get_all_odds([1,2,5]) ) # [1, 5]
print( get_all_odds([3,3,2,2,1,3]) ) # [3, 3, 1, 3]
odd_list = get_all_odds([3,3,2,2,1,3])
print(odd_list)
[3, 3, 1, 3]

Basic structure

▶ Create an empty answer list
▶ Examine each element in num_list
▶ If number is odd, append to answer
    answer.append(number)
▶ Return the answer list