CSC 110: Week 2, Lecture 2

Dr. Laura Bodine

Recap on numbers

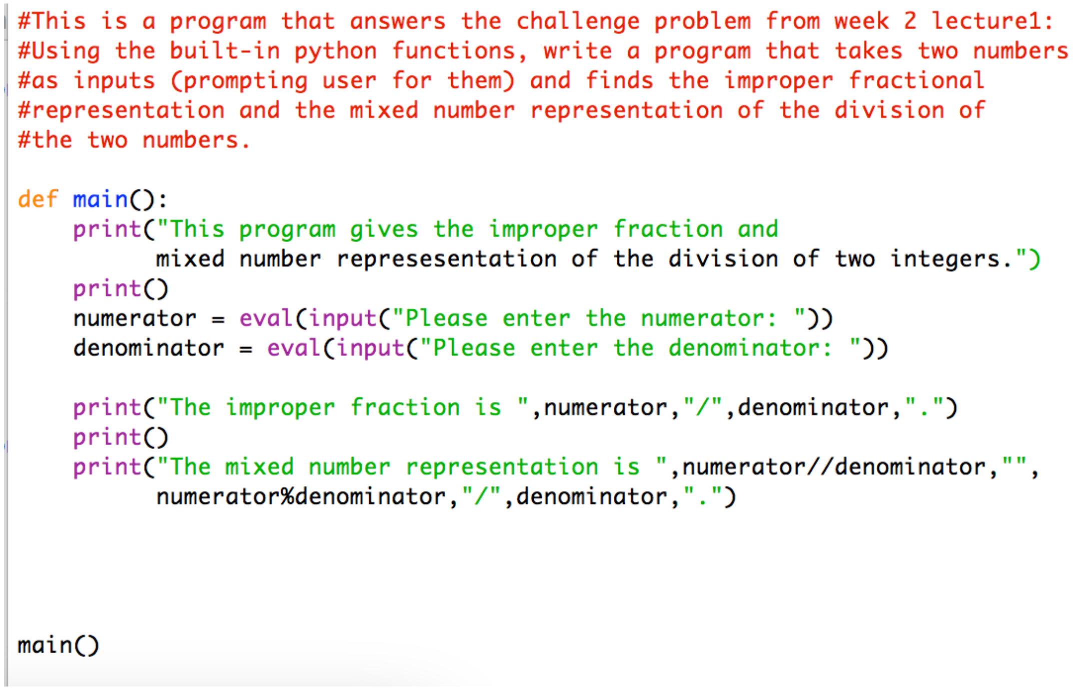
- Two basic numeric data types: ints and floats
  - Ints are whole numbers with no decimal point
  - Floats are numbers with decimal point (even if that decimal part is .0)
- Zero-based numbering
- Arithmetic
  - Built-in operations: *, /, +, -, **, //, %, abs
  - Single-type expressions
  - Mixed-type expressions
- Type conversion
  - Implicit and explicit

Challenge problem from last class

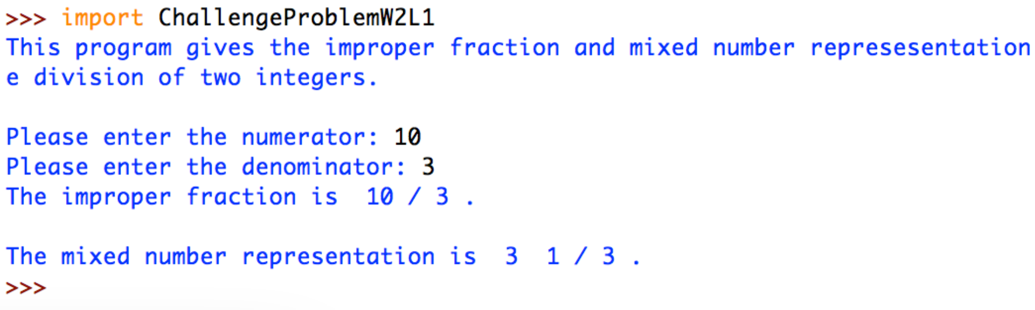
- Using the built-in python functions, write a program that takes two numbers as inputs (prompting user for them) and finds the improper fractional representation and the mixed number representation of the division of the two numbers.

- What do we need to use?
  - Print statements
  - Input statements
  - % and // operators

Challenge problem from last class



```
#This is a program that answers the challenge problem from week 2 lecture1:
#Using the built-in python functions, write a program that takes two numbers
#as inputs (prompting user for them) and finds the improper fractional
#representation and the mixed number representation of the division of
#the two numbers.

def main():
    print("This program gives the improper fraction and
          mixed number represesentation of the division of two integers.")
    print()
    numerator = eval(input("Please enter the numerator: "))
    denominator = eval(input("Please enter the denominator: "))

    print("The improper fraction is ",numerator,"/",denominator,".")
    print()
    print("The mixed number representation is ",numerator//denominator,"",
          numerator%denominator,"/",denominator,".")



main()
```



```
>>> import ChallengeProblemW2L1
This program gives the improper fraction and mixed number represesentation
e division of two integers.

Please enter the numerator: 10
Please enter the denominator: 3
The improper fraction is  10 / 3 .

The mixed number representation is  3  1 / 3 .
>>>
```

What we will cover today

- Math library
  - Access to math functions

- Range and lists of integers
  - Loops, accumulators

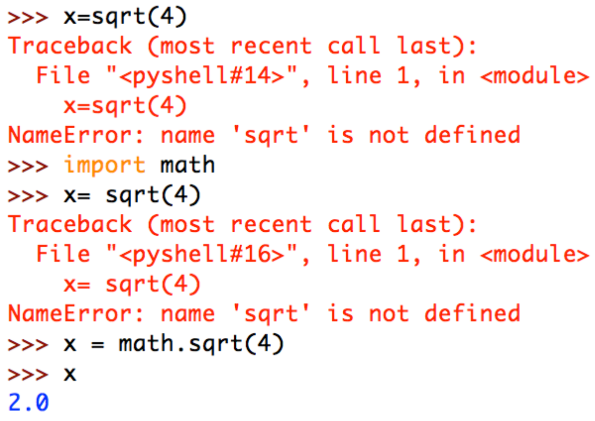
- Precision, conversion, rounding, etc…

Math Library!

- Built-in functions provide basic math
- More complicated math requires the math library
    - Broader functions and constants
        - pi
        - e
        - Square root
        - Trigonometric functions (sine, cosine, tangent, arcsin, arccos, arctan)
        - Logarithms
        - Exponentials
        - Ceiling
        - Floor

Importing the math library

- Use the standard import statement.
  - >>> import math
  - Use math functions with the dot notation: math.squareroot(<number>)



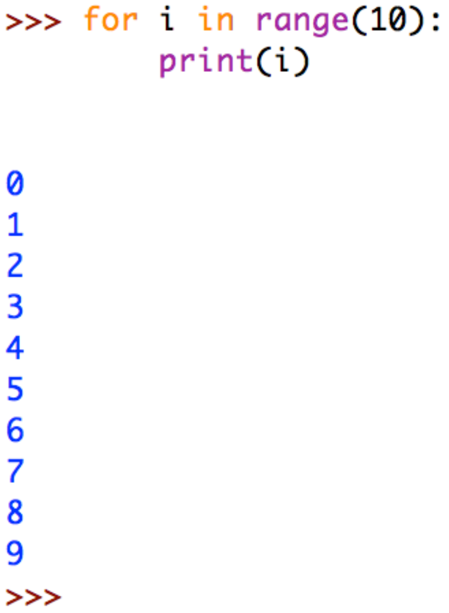
```
>>> x=sqrt(4)
Traceback (most recent call last):
  File "<pyshell#14>", line 1, in <module>
    x=sqrt(4)
NameError: name 'sqrt' is not defined
>>> import math
>>> x= sqrt(4)
Traceback (most recent call last):
  File "<pyshell#16>", line 1, in <module>
    x= sqrt(4)
NameError: name 'sqrt' is not defined
>>> x = math.sqrt(4)
>>> x
2.0
```

Let’s try it out

- Try to parse the following expressions in python
  - the square root of 121
  - $\log_2(64)$
  - $e^{\pi t}$, evaluted where t = 2
- What happens when we parse the expressions using variables and then change the variable assignment?

Looking at basic loops

- Last week the book introduced a loop using this range function



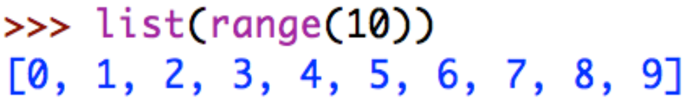
```
>>> for i in range(10):
        print(i)


0
1
2
3
4
5
6
7
8
9
>>>
```

range(<integer>)

- This range function is a way of generating a set of integers
- When would this be helpful?
- Can be used in conjunction with a list

list(range(<integer>))

- Built-in way to make numbered lists
  - list(range(<integer>)
  - Zero-based numbering!

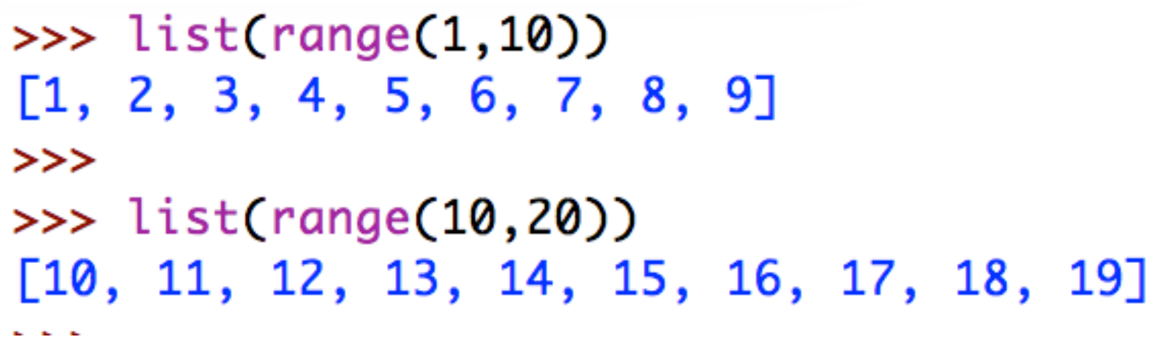


```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- The <integer> represents the number of elements in the list
  - Note: not equal to the largest value stored in the list. Why?

list(range(<integer>,<integer>))

- Built-in way to make lists that do not start at 0
  - Why might we want this to be the case?



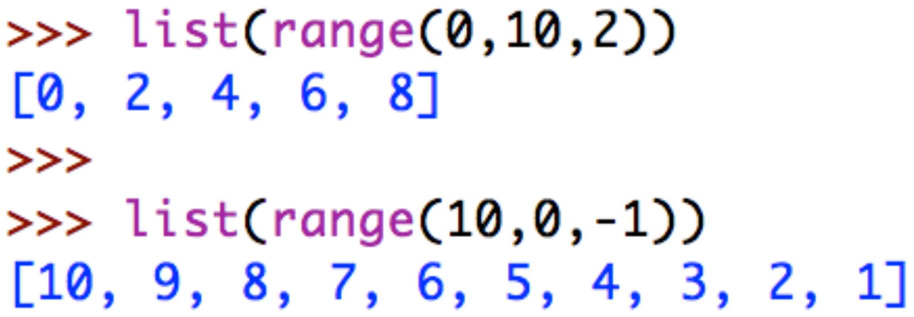
```
>>> list(range(1,10))
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>
>>> list(range(10,20))
[10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

Think-pair-share

- Think about the line: list(range(<integer>,<integer>))
  - What is the first argument?
  - What is second argument?
  - Does this change our understanding of the argument in list(range(<integer>))

list(range(<integer>,<integer>,<integer>))

- Built in way to make lists that don’t count by 1
  - Can make even lists or lists that count by 3
  - Can make lists that count backwards



```
>>> list(range(0,10,2))
[0, 2, 4, 6, 8]
>>>
>>> list(range(10,0,-1))
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

Let’s try it out

- Try making a list that starts at 15, counts by 5s up to an including 100
  - [15, 20, …100]

- Try making a loop that prints the multiples of 3 that are less than 63 and greater than 0

- Try making a list that prints from largest to smallest the numbers between 0 and 63 that are divisible by 3

Now, didn’t we say something about loops?

- May want to loop over a set and accumulate the results
  - Running tally of total, product of a set of numbers

- Example, I run a bookstore. My list stores the number sold for each title. I want the total number of books sold.
  - I want to loop over the elements in the list and add them up
  - Need a running total: accumulator variable

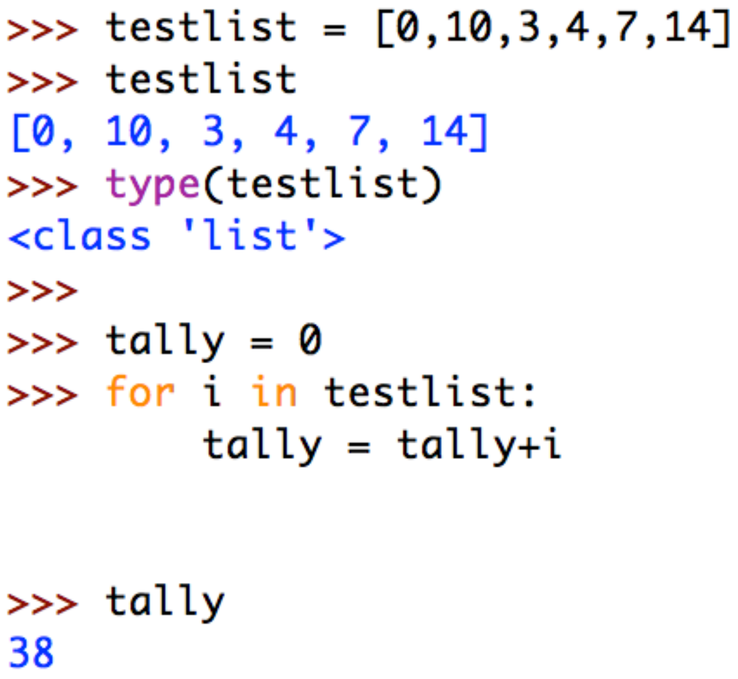
Accumulator loops

- Say my bookstore list is [0,10,3,4,7,14]

- Accumulator variable tally is assigned to 0 at the start

- Loop over elements in the list, adding them to tally

Sample loop



```
>>> testlist = [0,10,3,4,7,14]
>>> testlist
[0, 10, 3, 4, 7, 14]
>>> type(testlist)
<class 'list'>
>>> 
>>> tally = 0
>>> for i in testlist:
        tally = tally+i


>>> tally
38
```

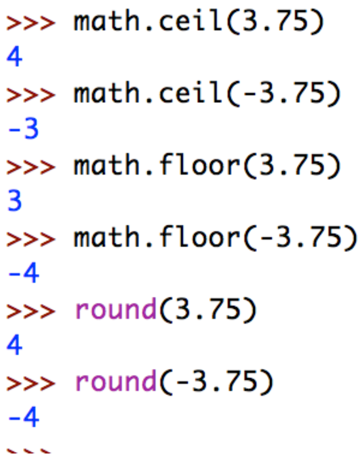
Precision in numeric types

- Ints are stored exactly
  - Fixed-size: 2^n values can be stored in n bits
  - What about long ints?

- Floats are stored as approximations
  - Can represent numbers larger than 2^n
  - Fixed precision

Ceiling, Floor, Round functions

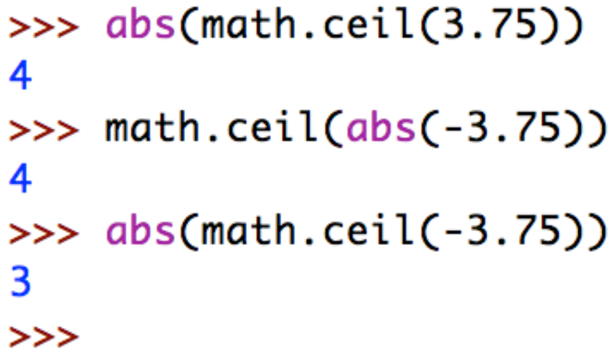
- Ceiling: nearest integer larger than the value
- Floor: nearest integer smaller than the value
- Round: the closest integer (if the decimal is a 0.5 round up)



```
>>> math.ceil(3.75)
4
>>> math.ceil(-3.75)
-3
>>> math.floor(3.75)
3
>>> math.floor(-3.75)
-4
>>> round(3.75)
4
>>> round(-3.75)
-4
```

Be careful!

- Negative arguments can get tricky!
- Be sure that you understand the order of operations to whatever mathematical idea you are trying to implement



```
>>> abs(math.ceil(3.75))
4
>>> math.ceil(abs(-3.75))
4
>>> abs(math.ceil(-3.75))
3
>>>
```

Sample problems from the book

- Chapter 3 Q2
  - Write a program that calculates the cost per square inch of a circular pizza given its diameter and price

- Chapter 3, Q 6
  - Two points in a plane are specified using the coordinates (x1,y1) and (x2,y2). Write a program that calculates the slope of a line through two (non-vertical) points entered by the user:
    - Slope = (y2-y1)/(x2-x1)