Last Time

integer/float/string

import math
math.sqrt()
math.pi
Extra Question from Lab 2

* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
Type Conversion

- In *mixed-typed expressions* Python will convert ints to floats
- Sometimes we want to control the type conversion: *explicit typing*
Type Conversions

>>> float(22//5)
4.0

>>> int(4.5)
4

>>> int(3.9)
3

>>> round(3.9)
4

>>> round(3) 
3
Using the Math Library

\[ x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \]

\text{discRoot} = \text{math.sqrt}(b*b - 4*a*c)
Using the Math Library

import math  # Makes the math library available.

def main():
    print("This program finds the real solutions to a quadratic")
    a, b, c = eval(input("Please enter the coefficients (a, b, c): "))

    discRoot = math.sqrt(b * b - 4 * a * c)
    root1 = (-b + discRoot) / (2 * a)
    root2 = (-b - discRoot) / (2 * a)

    print()
    print("The solutions are:", root1, root2)

main()
Using the Math Library

This program finds the real solutions to a quadratic
Please enter the coefficients (a, b, c): 3, 4, -1
The solutions are: 0.215250437022 -1.54858377035

• What do you suppose this means?

This program finds the real solutions to a quadratic
Please enter the coefficients (a, b, c): 1, 2, 3
Traceback (most recent call last):
  File "<pyshell#26>", line 1, in -toplevel-
    main()
...
    discRoot = math.sqrt(b * b - 4 * a * c)
ValueError: math domain error
>>>
Math Library

• a = 1, b = 2, c = 3
  – trying to take square root of negative number
Accumulating Results: Factorial

• Say you are waiting in a line with five other people. How many ways are there to arrange the six people?

• 720 (6!)

• Factorial is defined as:
  \[ n! = n(n-1)(n-2)\ldots(1) \]

• So, 6! = 6*5*4*3*2*1 = 720
Accumulating Results: Factorial

• How did we calculate 6!?
• 6*5 = 30
• Take that 30, and 30 * 4 = 120
• Take that 120, and 120 * 3 = 360
• Take that 360, and 360 * 2 = 720
• Take that 720, and 720 * 1 = 720

Aside: Interrobang

http://en.wikipedia.org/wiki/Interrobang
Accumulating Results: Factorial

• Doing repeated multiplications, **keeping track of the running product**

• **Accumulator algorithm**
  – we’re building up or **accumulating** the answer in a variable, known as the **accumulator variable**
The Software Development Process

• Analyze the Problem
• Determine Specification
• Create a Design
• Implement Design
• Test/Debug the Program
• Maintain Program
Accumulating Results: Factorial

• The general form of an accumulator algorithm looks like this:

- Initialize the accumulator variable
- Loop until final result is reached
  - Update value of accumulator variable
The Software Development Process

• Analyze the Problem
• Determine Specification
• Create a Design
• Implement Design
• Test/Debug the Program
• Maintain Program
Accumulating Results: Factorial

• It looks like we’ll need a loop!

```python
fact = 1
for factor in [6, 5, 4, 3, 2, 1]:
    fact = fact * factor
```

• Can trace through it to verify that it works
Accumulating Results: Factorial

• Why did we need to initialize fact to 1?
  – Each time through the loop, previous value of fact is used to calculate the next value of fact. By doing the initialization, you know fact will have a value the first time through.

• If you use fact without assigning it a value, what does Python do?
Accumulating Results: Factorial

• multiplication is associative and commutative
  – we can rewrite our program:

```python
fact = 1
for factor in [2, 3, 4, 5, 6]:
    fact = fact * factor
```

• Find the factorial of other numbers?
Accumulating Results: Factorial

- We can do the range for our loop a couple different ways.
  - We can count up from 2 to n:
    \[
    \text{range}(2, \ n+1)
    \]
  - We can count down from n to 2:
    \[
    \text{range}(n, \ 1, \ -1)
    \]
Accumulating Results: Factorial

Completed program:

```python
# factorial.py
# Program to compute the factorial of a number
# Illustrates for loop with an accumulator

def main():
    n = eval(input("Please enter a whole number: "))
    fact = 1
    for factor in range(n,1,-1):
        fact = fact * factor
    print("The factorial of", n, "is", fact)

main()
```

Python Programming, 2/e
Functions

• Passing data (used in new lab today)

def factorial(n):
    fact = 1
    for factor in range(n,1,-1):
        fact = fact * factor
    print("The factorial of", n, "is", fact)
Functions

• Getting data back (used in new lab today)

def factorial2(n):
    fact = 1
    for factor in range(n, 1, -1):
        fact = fact * factor
    return fact
• Write a function that sums numbers from n to m and returns the sum
  – Pass in n=3 and m=6, would return 18
    (because 3+4+5+6=18)

• Write a design
• Implement the design
The Limits of Int

• What is 100!?

```python
>>> main()
Please enter a whole number: 100
The factorial of 100 is
93326215443944152681699238856266700490715968264381621468592
96389521759999322991560894146397615651828625369792082722375
8251185210916864000000000000000000000000000000
```

• ?!
Python3 can handle it, but...

Python 1.5.2 (#0, Apr 13 1999, 10:51:12) [MSC 32 bit (Intel)] on win32
Copyright 1991-1995 Stichting Mathematisch Centrum, Amsterdam

>>> import fact
>>> fact.main()
Please enter a whole number: 13
13
12
11
10
9
8
7
6
5
4
Traceback (innermost last):
  File "<pyshell#1>", line 1, in?
    fact.main()
  File "C:\PROGRA~1\PYTHON~1.2\fact.py", line 5, in main
    fact=fact*factor
OverflowError: integer multiplication
The Limits of Int

• What’s going on?

  – Infinite number of integers but finite range of ints that can be represented

  – Range depends on the number of bits a particular CPU uses to represent an integer value
    • 32 bits is typical
The Limits of Int

• 32 bits: $2^{32}$ possible values, centered at 0
• Range then is $-2^{31}$ to $2^{31}-1$
  – need to subtract one from top end because of 0

• 100! is much larger than this...
Handling Large Numbers

• Does switching to float help?

• If we initialize the accumulator to 1.0, we get

```python
>>> main()
Please enter a whole number: 15
The factorial of 15 is 1.307674368e+012
```

• No longer get an exact answer
Handling Large Numbers: Long Int

- Very large and very small numbers are expressed in *scientific or exponential notation*

- \(1.307674368e+012\) means \(1.307674368 \times 10^{12}\)

- Decimal needs to be moved right 12 decimal places to get the original number, but there are only 9 digits
  - 3 digits of precision have been lost
Handling Large Numbers

- **Floats** are approximations
- **Floats** allow us to represent a larger range of values, but with lower precision.

- Python3 solution: expanding **ints**!
- Python3 **ints are not a fixed size**
  - expand to handle whatever value it holds
Handling Large Numbers

• Newer versions of Python automatically convert your ints to expanded form when they grow so large as to overflow.

• We get indefinitely large values (e.g. 100!) at the cost of speed and memory