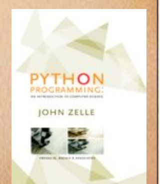
Python Programming: An Introduction to Computer Science

Chapter 3

Computing with Numbers

Updated by Dan Fleck



Objectives

- To understand the concept of data types.
- To be familiar with the basic numeric data types in Python.
- To understand the fundamental principles of how numbers are represented on a computer.
- To be familiar with variable scope

Numeric Data Types

- We've seen are two different kinds of numbers!
  - (5, 4, 3, 6) are whole numbers – they don't have a fractional part -- integers
  - (.25, .10, .05, .01) are decimal fractions - floating point

Numeric Data Types

  - Inside the computer, whole numbers and decimal fractions are represented quite differently!
  - We say that decimal fractions and whole numbers are two different *data types*.
- The data type of an object determines what values it can have and what operations can be performed on it.

Numeric Data Types: Integers

- Whole numbers are represented using the *integer* (*int* for short) data type.
- These values can be positive or negative whole numbers.

Numeric Data Types: Floating point

- Numbers that can have fractional parts are represented as *floating point* (or *float*) values.
- How can we tell which is which?
  - A numeric literal without a decimal point produces an int value
  - A literal that has a decimal point is represented by a float (even if the fractional part is 0)

Python's type function

- Python has a special function to tell us the data type of any value.

```
>>> type (3)
<type 'int'>
>>> type (3.1)
<type 'float'>
>>> type(3.0)
<type 'float'>
>>> myint = -32
>>> type(myint)
<type 'int'>
>>> myfloat = 32.0
>>> type(myfloat)
<type 'float'>
>>> mystery = myint * myfloat
>>> type(mystery)
<type 'float'>
```


Why two types?

- Why do we need two number types?
  - Values that represent counts can't be fractional (you can't have 3 ½ quarters)
  - Most mathematical algorithms are **very efficient** with **integers**
  - The float type stores only an ***approximation*** to the real number being represented!
  - Since floats aren't exact, **use an int whenever possible!**
  - **Lets check the speed with mathTimer.py!**

How operations work

- Operations on ints produce ints, operations on floats produce floats.

```
>>> 3.0+4.0
7.0
>>> 3+4
7
>>> 3.0*4.0
12.0
>>> 3*4
12
>>> 10.0/3.0
3.3333333333333335
>>> 10/3
3
>>> 10%3
1
>>> abs(5)
5
>>> abs(-3.5)
3.5
```


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 -- 720 is the factorial of 6 (abbreviated 6!)
- Factorial is defined as:
  n! = n(n-1)(n-2)…(1)
- So, 6! = 6*5*4*3*2*1 = 720

Accumulating Results: Factorial

- How we could we write a program to do this?
- Input number to take factorial of, n
  Compute factorial of n, fact
  Output fact

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

Exercise: Writing Factorial

- Okay, lets try it. To solve this problem we need to
  - Write the pseudocode
  - Test it (manually walk through it)
  - Write the python code
  - Let's do it!

Completed Factorial Program

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

def main():
    n = 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()
```


The Limits of Int

- What is 100! ?

>>> main()
Please enter a whole number: 100
The factorial of 100 is 93326215443944152681699238856266700490715968264381621468592963895217599993229915608941463976156518286253697920827223758251185210916864000000000000000000000000

- Wow! That's a pretty big number!
- An int in Python has a limit of 2^{32}-1 or 2147483647
- After that newer versions of Python will automatically convert to a Long Int (which can hold larger values)

How computers see "int"s

- What's going on?
  - While there are an infinite number of integers, there is a finite range of ints that can be represented.
  - This range depends on the number of *bits* a particular CPU uses to represent an integer value. Typical PCs use 32 bits.

Handling Large Numbers: Long Ints

- Floats are approximations
- Floats allow us to represent a larger range of values, but with lower precision.
- Python has a solution, the *long int*!
- Long Ints are not a fixed size and expand to handle whatever value it holds.

Handling Large Numbers: Long Ints

- To get a long int, put “L” on the end of a numeric literal.
- 5 is an int representation of five
- 5L is a long int representation of five

```
>>> 2L
2L
>>> 2L**31
2147483648L
>>> type(2L)
<type 'long'>
>>> 100000000000000000000000000000000000L + 25
100000000000000000000000000000000025L
```


Handling Large Numbers: Long Ints

- Calculations involving long int produce long int results.
- Newer versions of Python automatically convert your ints to long ints when they grow so large as to overflow.

```
>>> x = 2147483647
>>> x = x + 1
>>> x
2147483648L
>>> type (x)
<type 'long'>
>>> print x
2147483648
```


Handling Large Numbers: Long Ints

- We started out with x assigned the largest integer value, and then added 1.
- x was automatically changed to type long int.
- When we print long ints, the ‘L’ is dropped
- Why not use long ints all the time? – Less efficient, slow computations

Type Conversions

- We know that combining an int with an int produces an int, and combining a float with a float produces a float.
- What happens when you mix an int and float in an expression?
  x = 5.0 / 2
- What do you think should happen?

Type Conversions

- For Python to evaluate this expression, it must either convert 5.0 to 5 and do an integer division, or convert 2 to 2.0 and do a floating point division.
- Converting a float to an int will lose information
- Ints can be converted to floats by adding ".0"

Type Conversion

- In *mixed-typed expressions* Python will convert ints to floats.
- Sometimes we want to control the type conversion. This is called *explicit typing*.
- average = sum / n
- If the numbers to be averaged are 4, 5, 6, 7, then sum is 22 and n is 4, so sum/n is 5, not 5.5!

Type Conversions

- To fix this problem, tell Python to change one of the values to floating point:
  average = float(sum)/n
- We only need to convert the numerator because now Python will automatically convert the denominator.

Type Conversions

- Why doesn't this work?
  average = float(sum/n)
- sum = 22, n = 5, sum/n = 4, float(sum/n) = 4.0!
- Python also provides *int()*, and *long()* functions to convert numbers into ints and longs.

Type Conversions

```
>>> float(22/5)
4.0
>>> int(4.5)
4
>>> int(3.9)
3
>>> long(3.9)
3L
>>> float(int(3.9))
3.0
>>> int(float(3.9))
3
>>> int(float(3))
3
```


Type Conversions

- The *round* function returns a float, rounded to the nearest whole number.

```
>>> round(3.9)
4.0
>>> round(3)
3.0
>>> int(round(3.9))
4
```


If Statements (ch 7)

Most programs need to do different things depending on conditions. Decision structures solve this problem by allowing the program to "choose" different paths in different circumstances.

A boolean statement is one that evaluates to True or False (logical statement)

if <boolean statement> is true:
  run this code
elif <boolean statement> is true:
  run this code
Else:
  run this code

else and elif are Optional. Have none, either, or both

If Statements (ch 7)

if A is True:
    run codeA
elif B is True:
    run codeB
elif C is True:
    run codeC
else:
    run codeD

Truth Table

A	*B*	*C*	*Code*
T	*T*	*T*	
T	*T*	*F*	
T	*F*	*T*	
T	*F*	*F*	
F	*T*	*T*	
F	*T*	*F*	
F	*F*	*T*	
F	*F*	*F*	

Forms of if statements

```
def basic(temp):
    print "-" * 30 # Print 30 dashes
    print "Basic"
    if temp > 90: # Prints it is hot if the temp is over 90 degrees
        print "It is hot"
    print "End of Basic"

def usingElse(temp):
    print "-" * 30 # Print 30 dashes
    print "usingElse"
    if temp > 90:
        print "It is hot"
    else:
        print "It is NOT hot"
    print "End of usingElse"

def usingElif(temp):
    print "-" * 30 # Print 30 dashes
    print "usingElif"
    if temp > 90:
        print "It is hot"
    elif temp > 100: # "Else if" - can have as many as you want
        print "It is REALLY HOT!!"
    elif temp < 50: # "Else if" - can have as many as you want
        print "It is cold!"
    else:
        print "It is just right"
    print "End of usingElif"


def main():
    basic(100)
    basic(80)

    usingElse(100)
    usingElse(80)
```

What prints

What prints

What prints

What prints

Forms of if statements

```
def usingElse(temp):
    print "-" * 30 # Print 30 dashes
    print "usingElse"
    if temp > 90:
        print "It is hot"
    else:
        print "It is NOT hot"
    print "End of usingElse"

def usingElif(temp):
    print "-" * 30 # Print 30 dashes
    print "usingElif"
    if temp > 90:
        print "It is hot"
    elif temp > 100: # "Else if" - can have as many as you want
        print "It is REALLY HOT!!"
    elif temp < 50: # "Else if" - can have as many as you want
        print "It is cold!"
    else:
        print "It is just right"
    print "End of usingElif"


def main():
    basic(100)
    basic(80)

    usingElse(100)
    usingElse(80)

    usingElif(105)
    usingElif(80)
    usingElif(30)
    usingElif(90)

main()
```

What prints

What prints

What prints

What prints

Uh oh... how to fix that?

Boolean Logic

if (True):
    do something

and -- do a logical and
    both conditions must be true for the statement to be true

or -- do a logical or
    if either condition is true the statement is true

not -- logical not (negate the statement)
    if it is True, make it false
    if it is False, make it true

```
>>>
>>> True and True
True
>>> True and False
False
>>> False and True
False
>>> False and False
False
>>>
>>>
>>>
>>> True or True
True
>>> True or False
True
>>> False or True
True
>>> False or False
False
>>>
>>>
>>> not True
False
>>> not False
True
>>>
```

Boolean logic examples

```
if temp > 90 or temp < 30:
    print "It is uncomfortable!"

    if cmp(userChoice, "Y") == 0 or
      cmp(userChoice,"y") == 0 or
      cmp(userChoice,"yes") == 0 :
      print "User said yes"
    else:
      print "User said No"
```

This is just an example but there are much better ways to do this. How?

Lets talk about functions

- Coming up:
  - Variable Scope
  - Function parameters
  - Why use functions?

Variable Scope

- Every variable has a "scope".
- The *scope* of a variable refers to the places in a program a given variable can be referenced.
- Variables defined in a function are local variables and can only be referenced directly in that function
- Variables defined in the module (outside a function) can be access anywhere in that module. "global"

Scope Examples

See scopeExamples.py

Global Keyword

- The global keyword is used to tell Python NOT to overwrite the global variable with a local one. Instead, use the global value.

```
myVar = 'test'
def myFunction():
    global myVar
    myVar = 'not test'

print 'Step one: ',myVar
myFunction()
print 'Step two:', myVar
```

→

>>> Step one: test

>>> Step two: not test

Parameters

Now we can understand parameters better. When we call a function a copy of the parameter's value is passed in and assigned to a local variable

```
def myFunction(input1):
        input1 = input1 + 25
        print input1

tempVar = 5
myFunction(tempVar)
myFunction(20)
myFunction(tempVar+10)
```


Parameter Copies

When we change a value of a parameter inside the function, does the new value remain outside the function?

Answer: No, it is a copy… so the original remains the same.

Then how do I get a value out of a function?

Answer: The return statement!

Parameter and Return Examples

```
def parameterTest(in1, in2, myVar):
    in1 = in2 * 2
    myVar = 'some other value'

in1 = 5
in2 = 10
name = 'Dan'
parameterTest(in1, in2, name)
print 'Name=%s  In1=%d In2=%d", name, in1, in2)
    >>> Name=Dan In1=5 In2=10    # NO CHANGES!
```

Parameter and Return Examples 2

```
def parameterTest(in1, in2, myVar):
    in1 = in2 * 2
    myVar = 'some other value'
    return in1

in1 = 5
in2 = 10
name = 'Dan'
in1 = parameterTest(in1, in2, name)
print 'Name=%s  In1=%d In2=%d", name, in1, in2)
    >>> Name=Dan In1=20 In2=10  # in1 changed!
```


Parameter and Return Examples 3

```
def parameterTest(in1, in2, myVar):
    in1 = in2 * 2
    myVar = 'some other value'
    return in1, in2, myVar

in1 = 5
in2 = 10
name = 'Dan'
in1, in2, name = parameterTest(in1, in2, name)
print 'Name=%s  In1=%d In2=%d", name, in1, in2)
    >>> Name=some other value In1=20 In2=10
```

You can return multiple values using Python's multiple assignment

Why use Functions

- Having similar or identical code in more than one place has some drawbacks.
  - Issue one: writing the same code twice or more.
  - Issue two: This same code must be maintained in two separate places.
- Functions can be used to reduce code duplication and make programs more easily understood and maintained.

Functions Informally

- A function is like a *subprogram*, a small program inside of a program.
- The basic idea – we write a sequence of statements and then give that sequence a name. We can then execute this sequence at any time by referring to the name.

Function Example

We've seen many examples of functions:

Calling a function

X = math.sqrt(36)

This calls the square root function and sets X equal to the return value.

Defining a function

```
def square(x):
    val = x * x
    return val
```


Function Example (cont)

Defining a function

```
def square(x):
    val = x * x
    return val

Z = square(10)
```

X is a parameter

x is created when the function is called and initialized to the value passed into the function

Square is the name of the function

Val is a local variable

Local variables only exist in the function. After the function returns you cannot use val anymore.

Functions as a design tool

- When writing code you first write pseudocode, then the real code.
- Using functions you can "design" the code and then fill in the details.
- For example: If you want to write a program to calculate interest on a home loan and display a bar graph.

Pseudocode

For example: If you want to write a program to calculate interest on a home loan and display a bar graph.

- Ask user for loan terms
- Calculate interest
- Draw bar chart

Pseudocode --> Real Code

```
def main():
    #Ask user for loan terms
    (years, principal, intRate) = getLoanTerms()
    # Calculate interest
    Interest = calculateInterest(years, principal, intRate)
    # Draw bar chart
    drawBarChart(interest, principal)
```

Now you need to fill in the details of the functions, but your "main" program is very clear and easy to understand. You have also broken down your problem into smaller chunks.

Example: Formatting a paragraph

- Lets say we're given a long string of text (maybe an entire newspaper article). We want to format this for reading on a small device, so we want to format it into 40 columns wide
- For example:
  Sen. McCain still has some official competition from former Arkansas Gov. Mike Huckabee, but his more pressing problem is the disdain that many conservatives feel toward him. To woo some on the right, the Arizona senator delivered a speech yesterday to the Conservative Political Action Committee in Washington, in which he drew distinctions between himself and the Democrats, the Washington Post says. On foreign policy, Sen. McCain criticized Sens. Hillary Clinton and Barack Obama for not recognizing the threat posed by Iran's nuclear ambitions. "I intend to defeat that threat by staying on offense," the Post quotes him as saying.

Step 1: Formatting a paragraph

- Step 1: Write out (in english) how would you do this on paper?
- If that's not clear, go ahead and start doing it for the paragraph below… what are you doing?

- Sen. McCain still has some official competition from former Arkansas Gov. Mike Huckabee, but his more pressing problem is the disdain that many conservatives feel toward him. To woo some on the right, the Arizona senator delivered a speech yesterday to the Conservative Political Action Committee in Washington, in which he drew distinctions between himself and the Democrats, the Washington Post says. On foreign policy, Sen. McCain criticized Sens. Hillary Clinton and Barack Obama for not recognizing the threat posed by Iran's nuclear ambitions. "I intend to defeat that threat by staying on offense," the Post quotes him as saying.

Step 2: Write that in pseudocode

English	Pseudocode
	strIndex = 0
Count 40 characters	Loop forever strIndex = Count 40 characters end loop if not 40 chars left
Backup up until we find the beginning of a word	strIndex = backup, until first space character
Add in newline	insertNewlineAt (strIndex)
Repeat from the next newline	

Step 2: Example

Pseudocode

strIndex = 0
Loop forever
    strIndex = Count 40 characters
    end loop if not 40 chars left
    strIndex = backup, until first
        space character
    insertNewlineAt (strIndex)

Char #40

Example

Personal firewall software may warn about the connection IDLE makes…

Backup and find this space

Personal firewall software may warn
about the connection IDLE makes…

Add a newline and you get this

Coming up: Step 3: The code we know

53

Step 3: The code we know

```
def reformatString(inputStr, cols):
    strIndex = 0  # Character number we're currently at in the String
    # this isn't great ---> for i in range(10000):
    while(True):
        strIndex += 40

        # Exit loop if there aren't 40 chars left
        if strIndex > len(inputStr):
            break

        # Backup and find the first space before strIndex
        firstSpace = How??

        # Insert a newline character (line break) there
        how?

    return inputStr
```

Coming up: Up next…

54

Up next…

- Find and understand the functions/syntax to finish solving this problem
- Questions?

Coming up: Up next…

55