The float type and more on variables

FEB 6TH 2012

The `float` type

- Numbers with decimal points are easily represented in binary:
  - 0.56 (in decimal) = 5/10 + 6/100
  - 0.1011 (in binary) = ½+0/4 + 1/8 +1/16
- The i^{th} bit after the decimal point has place value $1/2^i$.
- **Example:** 0.1101 = ½ + ¼ + 1/16 = 13/16 = 0.8125
- However, not all real numbers (even rational numbers) can be represented *exactly* by finite sums of these fractions.

Be wary of floating point errors

- Try 0.1 + 0.2
- Try adding 0.1 ten times.
- Try 0.1 + 0.1 + 0.1 – 0.3

- In general, *never* test for equality with floating point numbers.
- This is an infinite loop! Try it.

```
sum = 0.1
while sum != 1:
        sum = sum + 0.1
```

Some functions for floating point numbers

- The math module contains functions (e.g., `math.sqrt(x)`) for floating point numbers.

Function	What it does
`math.ceil(x)`	Returns the ceiling of x as a float
`math.floor(x)`	Returns the floor of x as a float
`math.trunc(x)`	Returns x truncated to an int
`math.exp(x)`	Returns e^x
`math.log(x)`	Returns logarithm of x to the base e
`math.log(x, b)`	Returns logarithm of x to the base b

There are many other functions in the math module: trignometric, hyperbolic, etc. There are also constants: math.pi and math.e.

Try solving these problems

- Given the radius of a circle, find its area.
- Given a positive integer, find the number of digits it has.

  Example: `int(math.ceil(math.log(565656, 10)))`
- There are also some built-in Python functions that are useful for math:
  - `round(x, n)`: returns the floating point value *x* rounded to *n* digits after the decimal point. If *n* is omitted, it defaults to zero.
  - `abs(x)`: returns the absolute value of *x*

Range of floating point numbers

- What is the largest floating point number in Python? Here is an interesting way to find out:

```
prod = 1.0
while prod*1.1 != prod:
        prev = prod
        prod = prod*1.1
print prev, prod
```

- The output is 1.78371873262e+308 inf

What does this output mean?

- Python uses an object called `inf` to represent positive infinity.
- When `1.78371873262e+308` was multiplied by 1.1 (i.e., increased by 10%), we went beyond the upper limits of type `float`.
- This means that the largest floating point number in Python has 308 digits.
- Notice that the `while`-loop terminated because `inf * 1.1` equals `inf`.

A better version of this program

```
import math
prod = 1.0
while not math.isinf(prod):
        prev = prod
        prod = prod*1.1
print prev, prod
```

- There is a function called `isinf(x)` in the `math` module that tells us if `x` equals `inf`.

The sys module contains information on the largest float

- Try:

```
import sys
sys.float_info.max
```

- On my machine this value is

  1.7976931348623157e+308

Sequence types

- There are seven sequence types in Python: *strings*, *Unicode strings*, *lists*, *tuples*, *bytearrays*, *buffers*, and *xrange* objects.
- Later we will study study strings, lists, and tuples in more detail.
- There are many powerful built-in operations on sequence types provided by Python. Stay tuned for details.

Variables in Python

- Variables are "sticky notes" attached to objects.
- What happens during the assignment statement?

```
x = 10
```

- A memory cell (made up of 4 or 8 bytes) is created and 10 is placed in it.

- The name `x` is attached ("stuck") to this memory cell.

More on variables

- What happens when `x = x + 1` is executed?

1. The object that x is attached to (i.e., 10) is copied into some working area.
2. 1 is added to this object.
3. The new object (i.e., 11) is moved into a (different) memory cell.
4. The name x is now attached to this new memory cell.

Multiple "sticky notes" at the same location

- What happens when we execute:

```
x = 10
y = x
x = x +1
```

1. x is a "sticky note" attached to a memory cell containing 5.
2. Then y is also stuck to this very location.
3. When x = x + 1 is executed, remember the memory cell containing 10 remains unchanged and the "sticky note" x is moved to the cell with 11.
4. Therefore y continues to have value 10.

Variable names

- Variable names need to start with a letter (upper or lower case) or an underscore (i.e., _).
- Following the first character, any sequence of letters, digits, and underscores is allowed.
- Python has a small number of *keywords*, that cannot be used as variable names:

and	del	from	not	while	as	elif	global
or	with	assert	else	if	pass	yield	break
except	import	print	class	exec	in	raise	continue
finally	is	return	def	for	lambda	try	

More on variables

- Case matters. The variables `count` and `Count` are different.
- Do not use lower case el ("l"), upper case oh ("O"), or upper case eye ("I") as single letter variable names. These are hard to distinguish from numerals 0 and 1 in some fonts.
- Use meaningful names: e.g., `factorBound`, `myUpperLimit`, `sequenceLength`, etc.
- Watch out for spelling errors in variable names.

Scope of a variable

- In Python there is no explicit variable declaration.
- In many languages (C, Java, etc.) variables have to be declared before they can be used.
- In programs in these languages, a variable comes into existence when it gets declared.
- In Python, a variable comes into existence when it is first assigned a value.
- The variable lives until the end of the program or until it is explicitly deleted using the `del` operator (this operator will become useful later).
- The scope of a variable is the portion of the program that the variable is in existence for.