Data types and variables
Bits, bytes, words

- A *bit* (short for binary digit) is the smallest unit in a computer.

- A *byte* is 8 bits; a *word* is 2 bytes (16 bits).

- The `int` type is Python uses *at least* 32 bits (4 bytes).

- The largest `int` value (on my Windows laptop) is $2^{31} - 1 = 2147483647$. And the smallest is $-2^{31} = -2147483648$.

- On my Linux desktop `int` uses 64 bits. So the largest value is $2^{63} - 1$ and the smallest is $2^{63}$. 
Playing with these notions

- Try
  ```python
  import sys
  sys.maxint
  ```

- Also try this
  ```python
  n = -37
  bin(n)
  n.bit_length()
  ```

- Try this also
  ```python
  type(sys.maxint+1)
  ```
A few words on `long` type

- Integers of type `long` can be arbitrarily large (or small). In other words, the type `long` provides infinite precision.
- A `long` constant can be explicitly specified by appending an `L` at the end of the integer. Try

  \[
  x = 875L \\
  \text{type}(x)
  \]

- Operations can be performed on a mix of `long` and `int` objects; the type of the answer will be the larger type, i.e., `long`. 
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 $i^{\text{th}}$ 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.

```python
sum = 0.1
while sum != 1:
    sum = sum + 0.1
```
The math module contains functions (e.g., `math.sqrt(x)`) for floating point numbers.

<table>
<thead>
<tr>
<th>Function</th>
<th>What it does</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>math.ceil(x)</code></td>
<td>Returns the ceiling of x as a float</td>
</tr>
<tr>
<td><code>math.floor(x)</code></td>
<td>Returns the floor of x as a float</td>
</tr>
<tr>
<td><code>math.trunc(x)</code></td>
<td>Returns the x truncated to an int</td>
</tr>
<tr>
<td><code>math.exp(x)</code></td>
<td>Returns $e^x$</td>
</tr>
<tr>
<td><code>math.log(x)</code></td>
<td>Returns logarithm of x to the base e</td>
</tr>
<tr>
<td><code>math.log(x, b)</code></td>
<td>Returns logarithm of x to the base b</td>
</tr>
</tbody>
</table>

There are many other functions in the math module: trigonometric, 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:** \( \text{int(math.ceil(math.log(565656, 10)))) } \)

- There are also some built-in Python functions that are useful for math:
  - \( \text{round}(x, n) \): returns the floating point value \( x \) rounded to \( n \) digits after the decimal point. If \( n \) is omitted, it defaults to zero.
  - \( \text{abs}(x) \): returns the absolute value of \( x \)
What is the largest floating point number in Python? Unfortunately, there is no `sys.maxfloat`. Here is an interesting way to find out:

```python
prod = 1.0
while prod*2.0 != prod:
    prev = prod
    prod = prod*2.0
print prev, prod
```

- Python uses an object called `inf` to represent positive infinity, with `inf + 1` and `inf*2.0` equal to `inf`.
- On my laptop it is roughly 8.98846567431e+307
There are seven sequence types in Python: *strings*, *Unicode strings*, *lists*, *tuples*, *bytearrays*, *buffers*, and *xrange* objects.

Later we will study strings, lists, and tuples in more detail.

There are many very powerful built-in operations on sequence types provided by Python. Stay tuned for details.
Variables are “sticky notes” attached to objects. What happens during the assignment statement:

\[ x = 10 \]

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

- The name \( x \) is attached 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 memory cell.
4. The name \( x \) is now attached to this new memory cell.
Play with the function \texttt{id}(x)

\begin{itemize}
  \item \texttt{id}(x) returns the “identity” of the object \texttt{x}.
  \item This is an \texttt{int} (or \texttt{long}) which is guaranteed to be unique and constant for this object during its lifetime.
  \item Two objects with non-overlapping lifetimes may have the same \texttt{id} value
\end{itemize}