A Second Look: constants, data types, variables, expressions,...
More in-depth discussion

Now that we have solved our second programming problem, let us revisit a bunch of topics:

- Data types
- Variables
- Expressions
- Key words
- Built-in functions
- Modules
- Control-flow statements
We have seen four data types thus far:

- int: -90, 8987
- float: 9.98, -3.54
- str: “hello”, “a”
- bool: True, False
Numeric data types

- Python supports four numeric data types:
  - *plain integers*,
  - *long integers*,
  - *floating point numbers*, and
  - *complex numbers*.

- Plain integers, i.e., objects of type `int`, are those that fit in 32 bits or 64 bits (depending on the operating system).
A bit (short for binary digit) is the smallest unit of storage in a computer.

A byte is 8 bits.

Depending on the operating system on your machine, an int type in Python may be stored:
- in 4 bytes (or 32 bits) or
- in 8 bytes (or 64 bits).
Exploring the limits of the int type

- The `sys` module contains information about the largest possible integer on your machine.

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

- On my machine this showed me `9223372036854775807`

- Why? To find out, let us look at the binary equivalent of this number. Try:
  ```python
  x = sys.maxint
  bin(x)
  ```

- Note: `bin(x)` is a built-in Python function that returns the binary equivalent of a given integer. This is similar to the first Python program we wrote.
Exploring the limits of the int type

- On my machine the binary equivalent of `sys.maxint` is:
  
  `0b111111111111111111111111111111111111111111111111111111111111111`

- The “0b” at the beginning of the string is Python’s way of indicating that this is a binary string.

- The “0b” is followed by 63 1’s. This tells me that my machine is using 8 bytes (64 bits) to store objects of type int.

- Thus the largest possible int object is
  
  $$2^0 + 2^1 + 2^2 + ... + 2^{62} = 2^{63} - 1 = 9223372036854775807$$
Beyond the range of int

- The range of values that a variable of type `int` can take is from 
  \(-\text{sys.maxint} + 1\) to \text{sys.maxint}.

- The slight asymmetry between the lower limit and the upper limit is due to 
  the way negative numbers are represented in binary in computers.

- What would happen if you tried?
  
  ```python
  x = \text{sys.maxint}
  x = x + 1
  ```

- In many programming languages this would cause `x` to take on weird 
  values and this situation is called an `integer overflow`.

- But, Python has a very nice way of handling this situation!
The long type

- Python provides a type called `long` that can be used to represent integers that have arbitrarily large magnitude.
- If you tried:
  ```python
  x = sys.maxint
  x = x + 1
  ```
  the type of the variable `x` would automatically change from `int` to `long`, as soon its value exceeded the `int` upper limit.
- The programmer would not notice any difference because this type change would just happen behind the scenes.
A few words on long type

- A long constant can be explicitly specified by appending an `L` at the end of the integer. Try:
  
  ```
  x = 875L
  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`. Try:
  
  ```
  x = 100 + 200L
  y = long(10) + 1000
  ```
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
  - Adding 0.1 ten times
  - 0.1 + 0.2 - 0.3 == 0.0
  - sum = 0.1
    - while sum != 1:
      - sum = sum + 0.1

- In general, never test for *equality* of floating point numbers; test for *closeness*.

- This is a major issue in graphics. Geometric primitives such as: are these three points on a line? need to be implemented carefully.
Range of float

- Try
  ```python
  import sys
  sys.float_info
  ```

- You will get lots of information on floating point numbers on your system.
  - largest floating point number
  - maximum representable power of 10
  - smallest positive number that can be represented
  - maximum number of digits after decimal point that might be correctly represented.

- To get the maximum floating point number use
  ```python
  sys.float_info.max
  ```
Our discussion has completely ignored a very important class of data types in Python called *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!