Binary, Hexadecimal, Decimal

The decimal number: 123 = 1 * 10 ^ 2 + 2 * 10 ^ 1 + 3 * 10 ^ 0

The hexadecimal number: *0x*123 = 1 * 16 ^ 2 + 2 * 16 ^ 1 + 3 * 16 ^ 0 = 291 in decimal

This hex number: *0x*bf42 = 11 * 16 ^ 3 + 15 * 16 ^ 2 + 4 * 16 ^ 1 + 2 * 16 ^ 0 = 48962

Just google Hexidecimal to Decimal converter and you will get numerous online number converters

Hexadecimal (also **base 16**, or **hex**) is a positional numeral system with a radix, or base, of 16

A hexadecimal number is represented with digits 0 - 9 followed by a - f to represent equivalent of 10, 11 , 12, 13, 14, 15 in base decimal

You might encounter hexadecimal numbers to encode color values. It’s also frequently used to show a binary dump of any data because it’s more concise than straight binary.

The binary number: *0b*1011 = 1 * 2 ^ 3 + 0 * 2 ^ 2 + 1 * 2 ^ 1 + 1 * 2 ^ 0 = 11 in decimal

All calculations in a computer are done in binary. Digital circuit logic depends on a signal being off or on; that is 0 or 1.

You might encounter binary or hexadecimal encoding when dealing with representation of communication protocols between computers and hardware devices.

Bytes, Words, Integers, Unsigned Integers, and Floats

A byte is 8 bits. It is the smallest addressable component of computer memory. Characters encoded using ASCII can be contained in a byte.

A word is the maximum number of bits that can hold a computer address. In the old-fashioned days that was 16 bits. Now-a-days, word size is either 32 or 64 bits but more frequently it’s 64 bits. The computer CPU’s registers can hold a word and the computer data and address buses can transfer a word at a time.

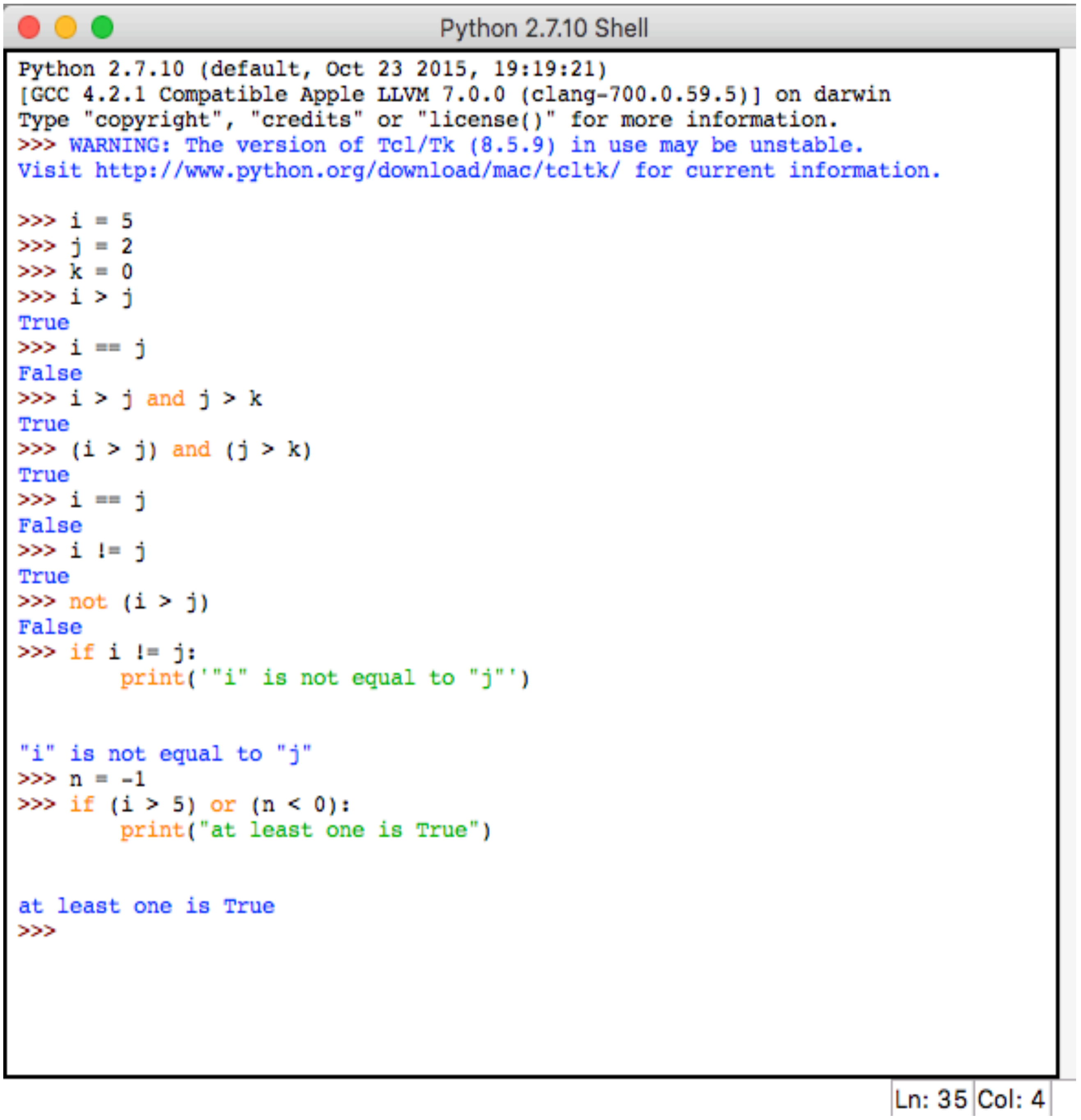
Demo of Binary, Hexadecimal and Ascii Conversions

```
>>> bin(23)
'0b10111'
>>> bin(23) and bin(1)
'0b1'
>>>
>>> int('0b10111',2)
23
>>> hex(163)
'0xa3'
>>> int('0xa3', 16)
163
>>>
>>> int('0xA3', 16)
163
>>> print("If either lower case or upper case is allowed, use lower
case")
If either lower case or upper case is allowed, use lower case
>>>
>>> chr(110)
'n'
>>> ord('n')
110
>>>
```

Demo of Binary and Boolean Operations

These are Python Idle sessions that demonstrate the difference between boolean and binary operations.

Figure 1. Boolean Operations



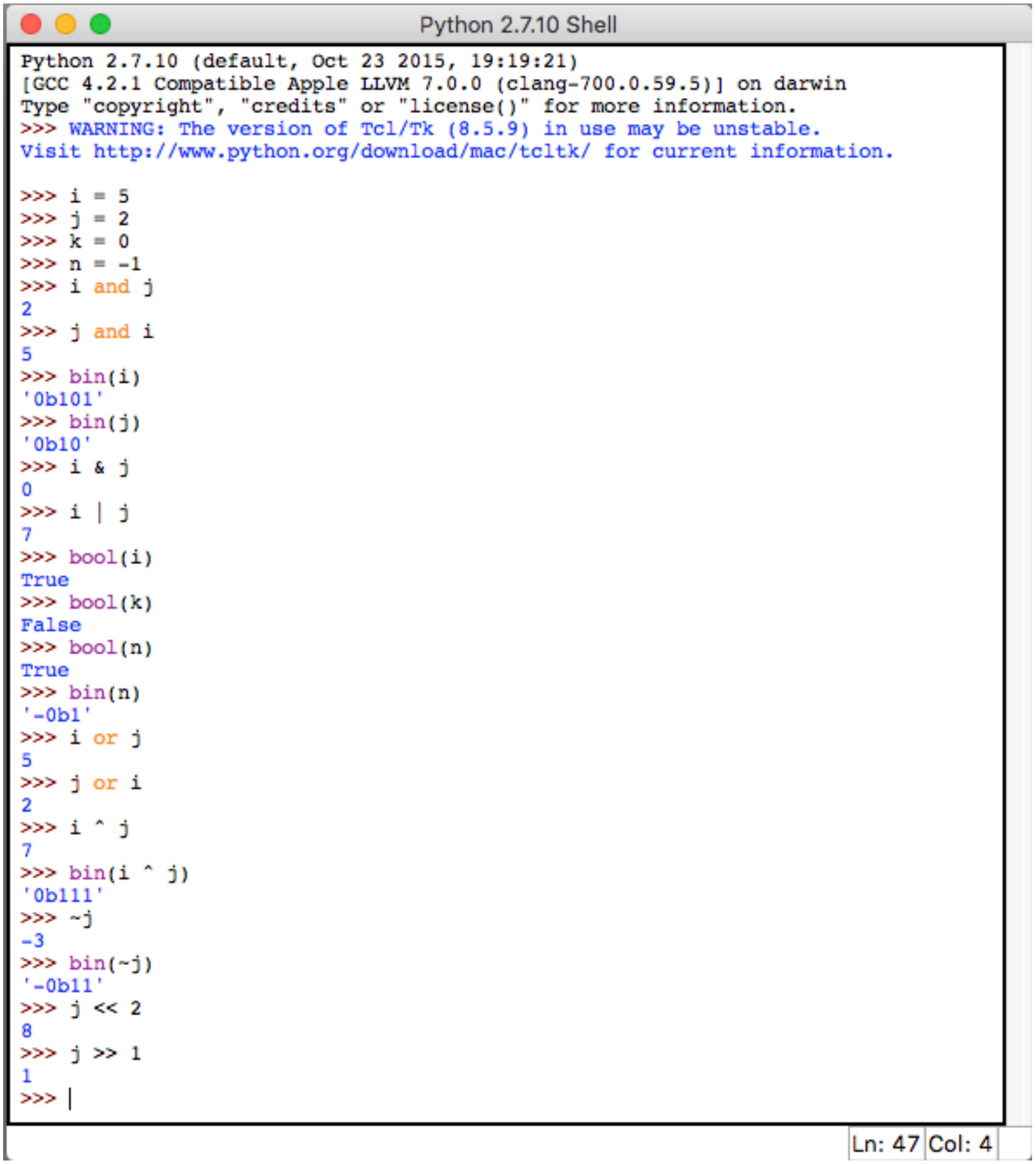
```
Python 2.7.10 (default, Oct 23 2015, 19:19:21)
[GCC 4.2.1 Compatible Apple LLVM 7.0.0 (clang-700.0.59.5)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> WARNING: The version of Tcl/Tk (8.5.9) in use may be unstable.
Visit http://www.python.org/download/mac/tcltk/ for current information.

>>> i = 5
>>> j = 2
>>> k = 0
>>> i > j
True
>>> i == j
False
>>> i > j and j > k
True
>>> (i > j) and (j > k)
True
>>> i == j
False
>>> i != j
True
>>> not (i > j)
False
>>> if i != j:
        print('"i" is not equal to "j"')


"i" is not equal to "j"
>>> n = -1
>>> if (i > 5) or (n < 0):
        print("at least one is True")


at least one is True
>>>
```

Figure 2. Boolean and Binary Operations



Python 2.7.10 Shell

```
Python 2.7.10 (default, Oct 23 2015, 19:19:21)
[GCC 4.2.1 Compatible Apple LLVM 7.0.0 (clang-700.0.59.5)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> WARNING: The version of Tcl/Tk (8.5.9) in use may be unstable.
Visit http://www.python.org/download/mac/tcltk/ for current information.

>>> i = 5
>>> j = 2
>>> k = 0
>>> n = -1
>>> i and j
2
>>> j and i
5
>>> bin(i)
'0b101'
>>> bin(j)
'0b10'
>>> i & j
0
>>> i | j
7
>>> bool(i)
True
>>> bool(k)
False
>>> bool(n)
True
>>> bin(n)
'-0b1'
>>> i or j
5
>>> j or i
2
>>> i ^ j
7
>>> bin(i ^ j)
'0b111'
>>> ~j
-3
>>> bin(~j)
'-0b11'
>>> j << 2
8
>>> j >> 1
1
>>> 
```