A picture containing drawing

Description automatically generated

**FAQ:**

1. Introduction to Python data types and objects
2. **Python Data types.**
3. **Declaring variables with assignment operators.**

**You cannot use reserved words or built-in identifiers** that have important purposes in Python, which you will learn about throughout this course. A list of python reserved words is described below. Creating names that are descriptive of the values often will help you avoid using any of these words. A quick table of these words is also available below.

A screenshot of a cell phone

Description automatically generated

1. **Numbers and Arithmetic Operators.**

FAQ: What do the operators <<, >>, &, |, ~, and ^ do?

These are Python's bitwise operators.

Preamble: Twos-Complement Numbers

All of these operators share something in common -- they are "bitwise" operators. That is, they operate on numbers (normally), but instead of treating that number as if it were a single value, they treat it as if it were a string of bits, written in twos-complement binary. A two's complement binary is same as the classical binary representation for positve integers but is slightly different for negative numbers. Negative numbers are represented by performing the [two's complement](http://en.wikipedia.org/wiki/Two's_complement) operation on their absolute value. So a brief summary of twos-complement binary is in order:

**Two's Complement binary for Positive Integers:**

* 0 is written as "0"
* 1 is written as "1"
* 2 is written as "10"
* 3 is "11"
* 4 is "100"
* 5 is "101"
* .
* .
* 1029 is "10000000101" == 2\*\*10 + 2\*\*2 + 2\*\*0 == 1024 + 4 + 1

**Two's Complement binary for Negative Integers:**

Negative numbers are written with a leading one instead of a leading zero. So if you are using only 8 bits for your twos-complement numbers, then you treat patterns from "00000000" to "01111111" as the whole numbers from 0 to 127, and reserve "1xxxxxxx" for writing negative numbers. A negative number, -x, is written using the bit pattern for (x-1) with all of the bits complemented (switched from 1 to 0 or 0 to 1). So -1 is complement(1 - 1) = complement(0) = "11111111", and -10 is complement(10 - 1) = complement(9) = complement("00001001") = "11110110". This means that negative numbers go all the way down to -128 ("10000000").

Of course, Python doesn't use 8-bit numbers. It USED to use however many bits were native to your machine, but since that was non-portable, it has recently switched to using an INFINITE number of bits. Thus the number -5 is treated by bitwise operators as if it were written "...1111111111111111111011".

Whew! With that preamble out of the way (and hey, you probably knew this already), the operators are easy to explain:

The Operators:

**x << y**

Returns x with the bits shifted to the left by y places (and new bits on the right-hand-side are zeros). This is the same as multiplying x by 2\*\*y.

**x >> y**

Returns x with the bits shifted to the right by y places. This is the same as //'ing x by 2\*\*y.

**x & y**

Does a "bitwise and". Each bit of the output is 1 if the corresponding bit of x AND of y is 1, otherwise it's 0.

**x | y**

Does a "bitwise or". Each bit of the output is 0 if the corresponding bit of x AND of y is 0, otherwise it's 1.

**~ x**

Returns the complement of x - the number you get by switching each 1 for a 0 and each 0 for a 1. This is the same as -x - 1.

**x ^ y**

Does a "bitwise exclusive or". Each bit of the output is the same as the corresponding bit in x if that bit in y is 0, and it's the complement of the bit in x if that bit in y is 1.

Just remember about that infinite series of 1 bits in a negative number, and these should all make sense.

**1. What is the difference between floating point and an integer?**

An integer has no decimals in it, a floating-point number can display digits past the decimal point.

**2. Why doesn't 0.1+0.2-0.3 equal 0.0?**

1This has to do with floating point accuracy and computer's abilities to represent numbers in memory. For a full breakdown, check out: https://docs.python.org/2/tutorial/floatingpoint.html

1. **Introduction to strings.**

**Are strings mutable? ----** Strings are not mutable! (meaning you cannot use indexing to change individual elements of a string)

**2. How do I create comments in my code? ----** You can use the hashtag # to create comments in your code.

1. Indexing and slicing.
2. String properties and methods.
3. **Print formatting with strings.**

Refer the link for more notes.

<https://pyformat.info/>

1. \*Split method in string.
2. Boolean, comparison operators.

**---X---**