Evaluating to True or False

Python has a number of operations that evaluate as one of the special values **True** or **False**. These include comparison operations, boolean operations, and object evaluations.

Comparison Operations

Comparison can be made either by value or identity. Comparing by value are more generalized than comparison by value. The comparison operators that compare by value are:

- **==** Equals
- **!=** Not Equals
- **<** Less Than
- **<=** Less Than or Equal
- **>** Greater Than
- **>=** Greater Than or Equal

In most cases, two objects that are of different types will always evaluate as not equal. So the comparison **1 == 'b'** will evaluate as **False**, and **1 != 'b'** will evaluate as **True**. One notable exception to this is numeric types, such as integers and floating-point numbers. The comparison **1 == 1.0** will evaluate to **True**, and **1 != 1.0** will evaluate to **False**.

Run the code below to see the results of some equality comparisons. You can also try changing the values and running it again.

```
1   print( 1 == 'b' )
2   print( 1 != 2 )
3   print( 1 == 1.0 )
4   print( 1 == '1' )
5   print( 1 != 2 )
```

```
False
True
True
False
True
```

The order comparisons, those that test greater or less than values, will generally raise an error if the objects compared are of different types. A notable exception, again, is numeric types. The comparison **3.0 >= 2** will result in True. The order of objects depends on the type of objects being considered. For example text (type string), uses lexicographic order and numeric types use numeric order.

Order comparison can be chained with multiple operators **1 < 2 <= 3** will result in **True**. Try running the examples below:

```
1  print( 1 < 33 )
2  print( 2 >= 2 )
3  print( 3.0 > 0 )
4  print( 'z' >= 'a' )
5  print( 1 <= 1.0 < 4 )
```

Run

Rese

```
True
True
True
True
True
```

The comparison operators which compare by identity are `is` and `is not`. They are most commonly used to compare against the special object `None`.

```
1  print( 1 is None )
2  print( True is not None )
```

Run

Rese

```
False
True
```

Membership Operations

Some objects in Python can contain others. For example, the word **"Henry"** (of type string), contains the letter **"r"** (also a string). The **in** operator tests for this type of membership. The expression **"r" in "Henry"** will return **True**, and **"b" in "Henry"** will return **False**.

```
1  print('e' in 'Henry' )
2  print('a' not in 'Henry' )
```

Boolean Operations

Boolean operations are based on boolean math, which you may have learned in a mathematics or philosophy course. The operators are **and**, **or**, and **not**. The first two take two operands, the last one, one operand. The **and** operator returns **True** if both of its operands evaluate as **True** and **False** if either evaluates to **False**. The **or** operator evaluates to **True** if either of its operands evaluates as **True** and **False** if they are both **False**. The **not** operator returns **False** if its operand evaluates to **True** and **True** otherwise. You can make more complex logical operations by nesting boolean operations in parenthesis. The expression **(False and False) or (not False)** evaluates to **True**, as **not False** is **True**.

```
1   print( True and False )
2   print( True and True )
3   print( False and True )
4   print( not True )
5   print( (False and False) or (not False) )
```

```
False
True
False
False
True
```

Object Evaluations

All objects (everything) in Python evaluates as **True** or **False**. This means you can use them in the places where you would test for **True** or **False**, such as in Boolean operations. Generally, most Python objects evaluate as True. The exceptions are:

1. Numeric types that equal zero, such as **0**, or **0.0**.
2. The constants **False** and **None**.
3. Anything that has a length of zero. This includes the empty string, **""**.

```
1   print( 0 or 'a' )
2   print( 0 and 0.0 )
```

```
a
0
```