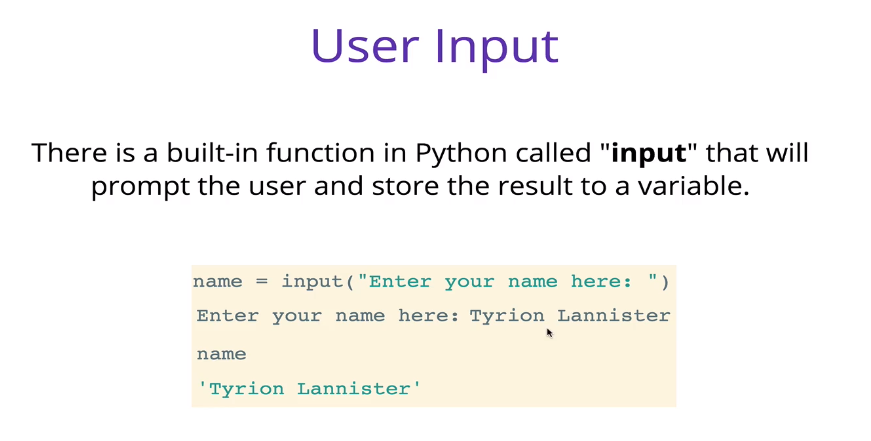
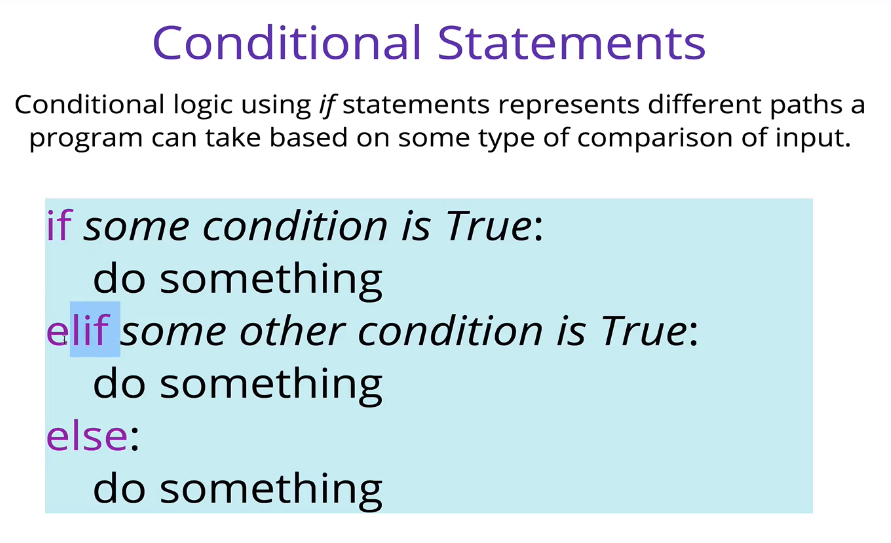
Section 8: Boolean and Conditional Logic

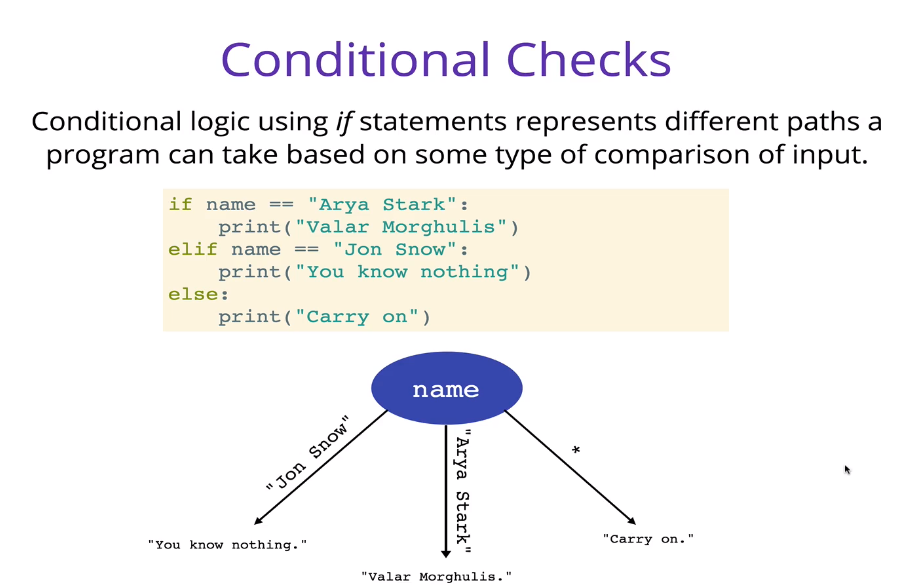
* Recall that Booleans are just True or False values. This section is about making decisions based on whether values are True or False
* Similarly, conditionals allow the program to make decisions based on whet her a condition is met (that is, whether a condition evaluates to True or False)
* User input!
  + This is important because we will make decisions based on what the user entesr



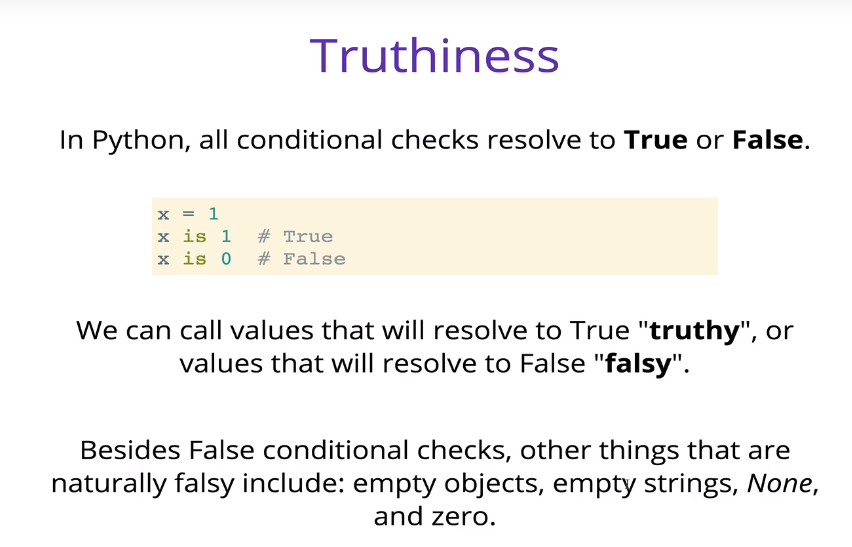
* + Remember that the result of an input is *always* a string
* Example file: input.py
* Conditional Statements



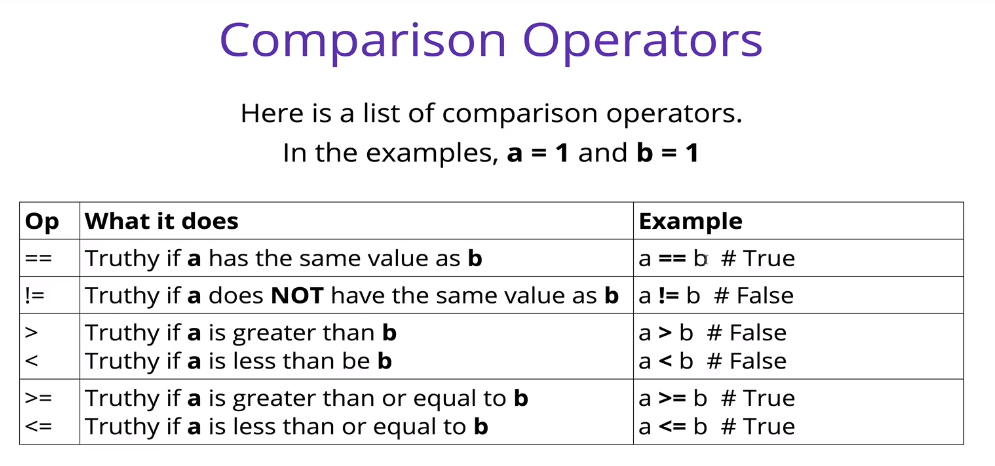
* + The code within an if or elif statement will only run if the condition is True
  + The colons at the end of the comparison statements is essential. Python will throw a syntax error if you do not use one
  + Else is a catch all that will only run if the “if” and “elifs” are all False
  + You don’t need to have an elif or even an else
  + You can have multiple elifs
  + In conditional statements, indentation matters!
    - The standard is four spaces for Python
* Conditional checks



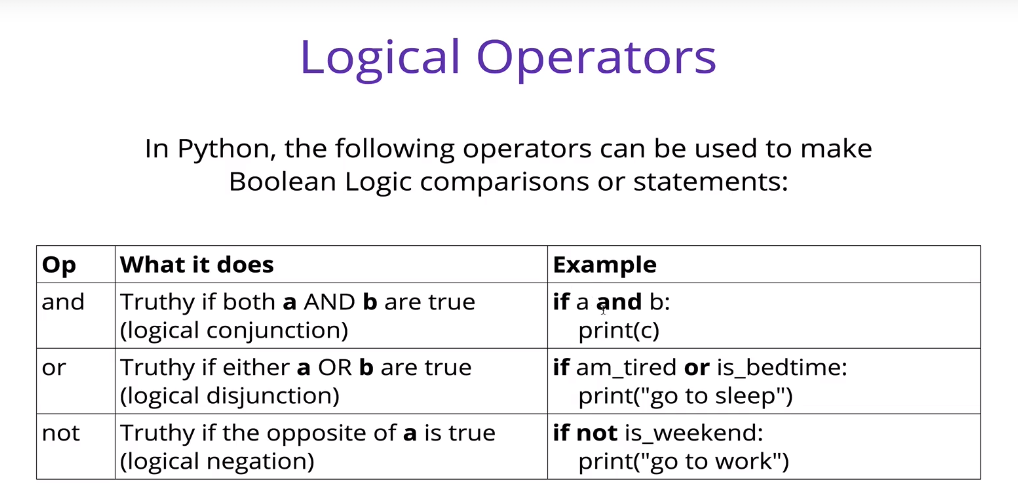
* + The double equal sign “==” is the check for equality in Python.
    - Remember that the single equal sign “=” is for assigning a variable
  + When comparing strings, everything matters:
    - Capitalization
    - Spacing
  + Example file: first\_Example.py
* Truthiness and Falsiness



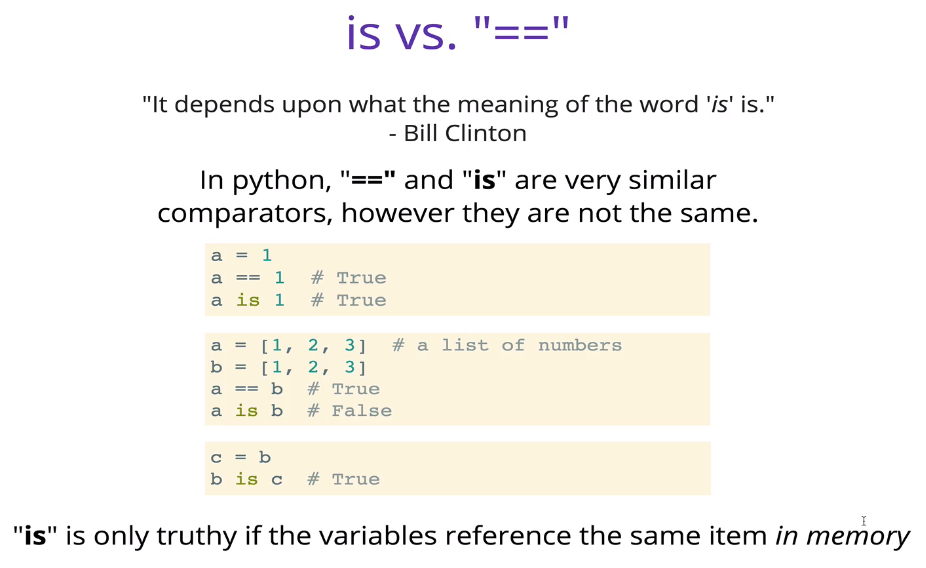
* + Some values are inherently Truthy or Falsy
    - 1 is inherently Truthy and 0 is inherently Falsy
* Comparison Operators
  + Comparison operators allow us to compare two pieces of data



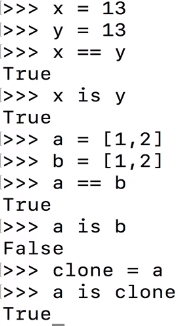
* + Comparing strings with > or < is inadvisable because it gets confusing
    - Python uses lexigraphical ordering, basically alphabetical ordering such as what the dictionary would use
    - Things get convoluted when you start comparing strings longer than one character
* Logical Operators
  + Logical operators connect comparison operators together to make more complex statements



* + and
    - This is known as a logical conjunction
    - For an “and” statement to resolve to true, all comparisons in the *and* statement must be True
    - If one or more are false, the statement will resolve
  + or
    - This is a logical disjunction operator
    - For this to resolve to true, at least one of the comparison statements must be true or Truthy.
    - If all are false or Falsy, the *or* statement will result to False
  + not
    - This is a logical negation
    - This statement resolves to True if the comparison is false (that is, “not a” is true if a is false)
    - Essentially, not will *negate* whatever the statement that comes after it evaluates to
  + Hint: when evaluating long comparison operators with many *ands* and *ors*, simply evaluate each comparison one at a time (put parentheses around them if it helps) and get it down to a series of Trues and Falses interspersed with *ands* and *ors*. That will make evaluating the statement much easier
* “is” vs “==”
  + “is” Is another test of equality, similar to but not the same as “==”
  + The main difference is that “==” compares values, whereas “is” tests whether two variables are stored in the **same place in memory**



* + In the following example:
    - “x == y” is True because they both refer to the same object (the number 13) in memory.
    - However, “a is b” is False because although they refer to lists that are equal to each other, those lists are separate objects in memory.
    - The statement “a is clone” is True because by setting *clone* equal to *a*, you have *clone* pointing to the same object in memory as *a*



* Bouncer Code-Along
  + Refer to bouncer.py