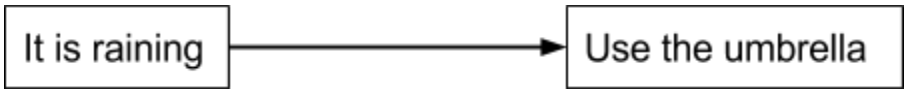
What is a Boolean?

A Boolean is any data that can only have one of two possible values. Generally, we use True or False. In a computer this True is represented as 1, and False is represented as 0. Booleans represent truth values in logical expressions. Booleans can be used in conditional statements, like ‘if’ and ‘if else’. The value of the Boolean (True or False) lets the program choose which code to execute. In code, Booleans most commonly take the form of a ‘comparator’, or ‘predicates’. Some examples include ‘==’ ‘>’ and ‘!=’, or any other function that returns True or False. These predicates can be thought of as questions that can only be answered with a Boolean, either True or False. Take the following example, (1 + 1) == 2. This predicate could be phrased as the question, “Is 1 + 1 equal to 2?”. The answer to this question may be fairly obvious to us, but a computer has no intuition. These predicates have to guide every decision a computer makes. Because these are so fundamental, an effective programmer must be able to construct and comprehend predicates.

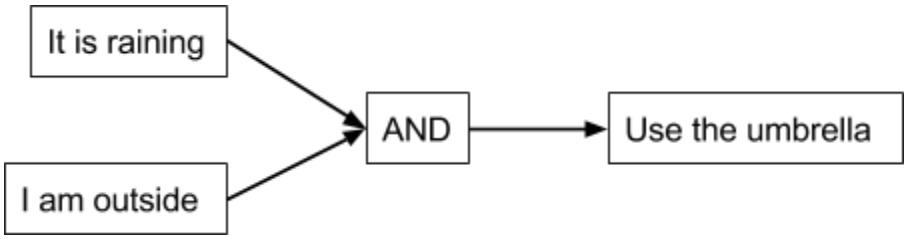
Writing Predicates with Bob.

As you may already have seen, boolean expressions are not limited to a single predicate. Many booleans can be nested together to create more complex boolean expressions. Consider the following scenario:

Bob is trying to figure out when it’s appropriate to use an umbrella. His first consideration is simple. IF ‘it is raining’ THEN ‘use an umbrella’.



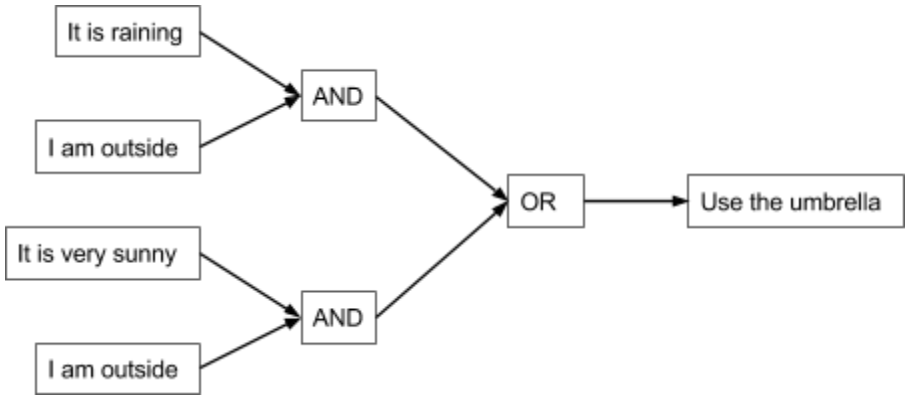
However, as rain splashed against the window and Bob sat on his couch with an umbrella over his head, he realized that his rule may have been too simple. It may be raining, but the umbrella serves no purpose indoors! Bob decides to add another condition (boolean) to his rule. ‘If it is raining AND I am outside, use an umbrella’.



But, as the torrential rain stopped and the bright summer sun returned, Bob began to resent his Transylvanian heritage, and saw another use for his umbrella. Bob decided to update his rule.

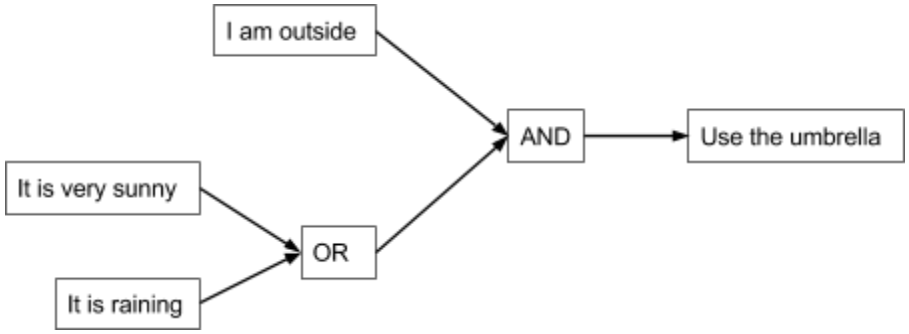
Bob correctly noticed that he should not use another AND to add the ‘Is it very sunny?’ Boolean. It probably won’t be both ‘very sunny’ and ‘raining’ at the same time. So, using another AND in the boolean expression would always return False. (Does that make sense?) Instead, Bob needs a boolean operator that can expand the number of possibilities which require an umbrella. Now bob sees **two** separate possibilities for the umbrella. Either Bob is outside and it is raining, OR Bob is outside and it is very sunny. Written more formally, we create the following boolean expression:

IF (‘It is raining’ AND I is outside’) OR (‘It is very sunny’ AND I is outside’)
THEN ‘Use the umbrella’



This rule works really well for Bob! But, being a Computer Scientist, Bob wants to re-write his boolean expression to be more simplified. ‘Bob is outside’ is a boolean on both sides of OR, thus if the boolean is True on the left, it will also be True on the right, and vise versa. Therefore, if ‘Bob is outside’ is False, the whole boolean expression is False [Remember: (False AND True) => False]. But if ‘Bob is outside’ is **True**, then Bob will use the umbrella if ‘It is raining’ OR ‘It is very sunny’. This is summarized in the expression below:

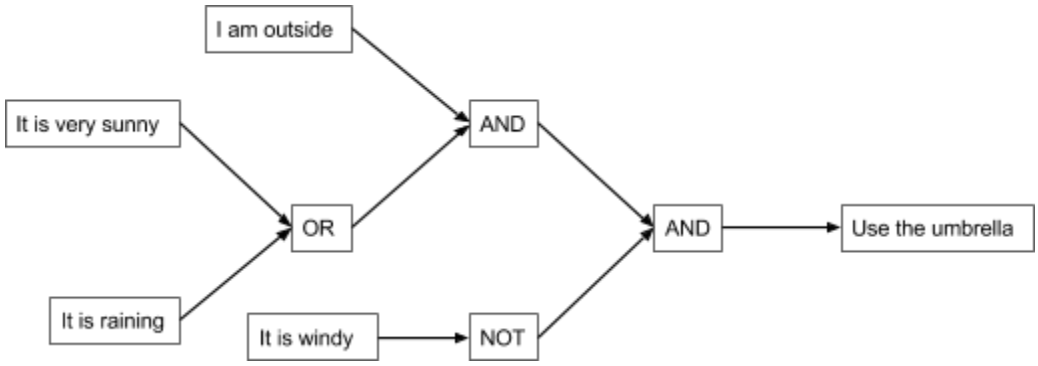
IF ‘I is outside’ AND (‘It is raining’ OR ‘It is very sunny’)
THEN ‘Use the umbrella’



Can you see how both of these boolean expressions represent the same rule? Try This: Test out a few different inputs. You'll notice both expressions return the same output for a given input. Confidently, Bob walked outside under his umbrella, into what could either have been rain or strong sunshine.

As Bob walked, the light breeze became stronger. A gust of wind caught Bob off guard and tore the umbrella out of his hand. Bob was furious. He would have to update his rule one more time. To eliminate this problem, Bob should only consider his umbrella when it is NOT windy. This flips a predicate, letting us answer the OPPOSITE of the original statement. When 'It is windy' is **True**, then NOT 'It is windy' will be **False**. The complete boolean expression follows:

IF **(**'Bob is outside' AND ('It is raining' OR 'It is very sunny') **)** AND (NOT 'It is windy')
THEN 'Use the umbrella'



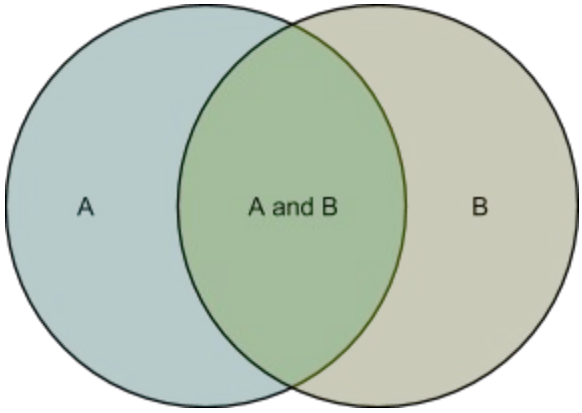
Bob was satisfied… At least for now. predicates can be combined using Boolean operators like AND, OR, & NOT to make a expressions which can answer almost any question. As Bob notices more factors that affect his decision, his boolean expression grows. Try This: Come up with another 'predicate' that would affect whether or not Bob decides to use his umbrella. Then modify the flow chart above to correctly include the predicate. (Examples: Temperature less than 32˚F, see-through umbrella, Bob likes piña coladas and gettin caught in the rain)

Basic Boolean Operators

AND

The Boolean operator, AND, only returns True when predicates on the left and right side of the AND are both True. If either predicate is False, AND will also return false. When both predicates are False, AND will return False. This operator is used to refine (or limit) the domain of inputs that return True.

Input 1	Input 2	Output of AND
T	T	**True**
T	F	False
F	T	False
F	F	False

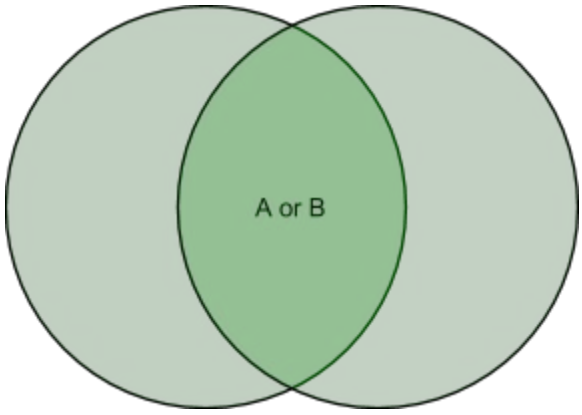


GRAPHIC: VENN DIAGRAM

OR

The Boolean operator, OR, returns True if EITHER predicate on the left or right side of the OR are True. If both are True, OR will also return True. This operator is used to expand the the domain of inputs that return True.

Input 1	Input 2	Output of OR
T	T	**True**
T	F	**True**
F	T	**True**
F	F	False



GRAPHIC: OR VENN DIAGRAM

NOT

The Boolean operator, NOT, returns the opposite of a predicate. Unlike OR and AND, NOT only takes one predicate as an argument. This is useful as a way of excluding certain inputs from returning True.

Input	Output of NOT
T	False
F	**True**