Booleans and Dictionaries

Theresa Migler-VonDollen
CMPS 5P

Computing with Booleans

- Boolean expressions evaluate to `True` or `False`.
- We have already used Boolean expressions to compare two values:

  ```
  while x > 0
  ```
- There are 3 Boolean operators:
  - `and`
  - `or`
  - `not`

Computing with Booleans

- The Boolean operators `and` and `or` combine two Boolean expressions to produce a Boolean result.
- Practice.
- The `and` of two expressions is true exactly when both of the expressions are true.
- We can display this in a *truth table*.

Truth Tables

- In a simple truth table there are two simple Booleans, P and Q.
- Since each expression has two possible values, there are four possible combinations of values.
- Practice with a simple truth table for `P and Q`.
- Practice with a simple truth table for `P or Q`.
- Practice with a simple truth table for `not P`.

Truth Tables

There are often times that we want to compute the value of Boolean operators on multiple Booleans:

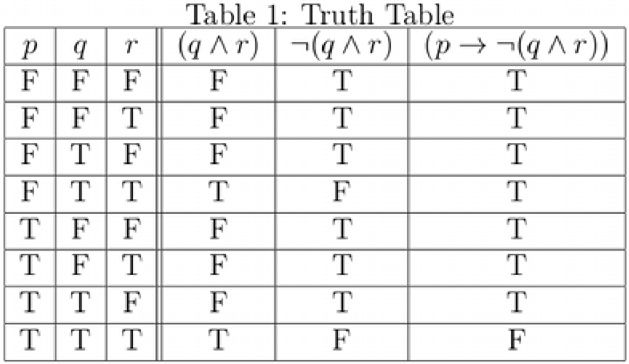

Table 1: Truth Table

p	q	r	$(q \wedge r)$	$\neg(q \wedge r)$	$(p \rightarrow \neg(q \wedge r))$
F	F	F	F	T	T
F	F	T	F	T	T
F	T	F	F	T	T
F	T	T	T	F	T
T	F	F	F	T	T
T	F	T	F	T	T
T	T	F	F	T	T
T	T	T	T	F	F

Boolean Operators

- Consider `a or not b and c`.
- How should this be calculated? What is the order of operations?
- The order of precedence from high to low is: `not`, `and`, `or`.
- `a or not b and c` is equivalent to `(a or ((not b) and c))`.
- It's best in practice to always use parentheses with Boolean operators.

Boolean Algebra

- `and` has similar properties to multiplication.
- `or` has similar properties to addition.
- `0` and `1` correspond to false.

DeMorgan's Laws

- `not (a or b) == (not a) and (not b)`
- `not (a and b) == (not a) or (not b)`
- It may be easier to figure out when a loop should stop, rather than when a loop should continue.
- In this case, write the loop termination condition and put a `not` in front of it. After a couple applications of DeMorgan's law you are ready to go with a simpler but equivalent expression.

Dictionaries

- A *dictionary* is a built in Python data type.
- Dictionaries are sometimes found in other languages as “associative memories” or “associative arrays”.
- Unlike sequences, which are indexed by a range of numbers, dictionaries are indexed by *keys*, which can be any immutable type; strings and numbers can always be keys.
- A dictionary is an unordered set of key: value pairs, with the requirement that the keys are unique (within one dictionary).

Dictionaries - How to Use

- A pair of braces creates an empty dictionary: `{}`.
- Placing a comma-separated list of key:value pairs within the braces adds initial key:value pairs to the dictionary; this is also the way dictionaries are written on output.
- The main operations on a dictionary are storing a value with some key and extracting the value given the key.
- The `keys()` method of a dictionary object returns a list of all the keys used in the dictionary, in arbitrary order.
- To check whether a single key is in the dictionary, use the `in` keyword.