Logic and Booleans

Introduction to Computer Programming

Dr. Paul Vrbik

September 16, 2018

Logical Operations and Constants

The following are the basic operands of logic.

1. `True`,
2. `False`,
3. `or`,
4. `and`,
5. `not`.

When we introduce loops we will also look at

$$\forall \qquad \exists$$

When we say $x + y$ this can also be interpreted as $+(x, y)$ where $+$ is taken to be the function

$$
\begin{aligned}
+ : \mathbb{Z} \times \mathbb{Z} &\to \mathbb{Z} \\
(x, y) &\mapsto x \text{ ``plus'' } y
\end{aligned}
$$

In general, any function which maps two inputs to one like

$$
\begin{aligned}
\oplus : A \times B &\to C \\
(a, b) &\mapsto c
\end{aligned}
$$

is called a binary function and can employ the short form

$$\oplus(a, b) = c = a \oplus b.$$

Definition (Boolean doman)

Let $\mathbb{B}$ denote the boolean domain where

$$\mathbb{B} = \{ \texttt{True} , \texttt{False} \} .$$

Definition (Predicate)

Any function that maps into $\mathbb{B}$ is called a predicate. (I.e. any function that evaluates to `True` or `False`.)

Example

A unary predicate test for prime testing:

$$P : \mathbb{Z} \to \{ \texttt{True} , \texttt{False} \}$$

$$x \mapsto \begin{cases} \texttt{True} & \text{if } x \text{ is prime} \\ \texttt{False} & \text{otherwise} \end{cases}$$

evaluates to true only when x is a prime number:

$$P(7) == \texttt{True} \qquad P(8) == \texttt{False} \qquad P(101) == \texttt{True} .$$

Note: Primality testing super hard!

If you like, write a Python program that implements this and try it on very large (64-bit) primes like $1\,319\,736\,134\,268\,565\,207$.

Example

The boolean function greater than

$$>: \mathbb{Z} \times \mathbb{Z} \to \mathbb{B}$$

is a predicate.

Question

Evaluate the following:

1. $>(3, 7)$,
2. $7 > 3$.

Definition (And)

The binary predicate ‘and’ is used to express that both of two statements are true and is false if either is false.

$$\texttt{and} \; : \mathbb{B} \times \mathbb{B} \to \mathbb{B}.$$

`and`	`True`	`False`
`True`	`True`	`False`
`False`	`False`	`False`

Definition (Or)

The binary predicate ‘or’ is used to express that at least one of two statements is true and is false only when both are false.

$$\texttt{or} \ : \mathbb{B} \times \mathbb{B} \to \mathbb{B}$$

`or`	`True`	`False`
`True`	`True`	`True`
`False`	`True`	`False`

```
>>> 1 > 0
True
>>> type(True)
bool

>>> True or False
True

>>> False or False
False
```

Question

Trace the following.

```
>>> 3>7 or 7>3
True
>>> 3>7 and 7>3
False
>>> 3>=3 and 7>=7
True
>>> 3>=3 and 7>7
False
```

Example

Consider that

$$(\ \texttt{True}\ \text{or}\ \texttt{False}\ )\text{and}\ \texttt{False}\ == \texttt{True}$$

$$\texttt{True}\ \text{or}\ (\ \texttt{False}\ \text{and}\ \texttt{False}\ ) == \texttt{False}\ .$$

is ambiguous and thus an order of operations is necessary.

Definition (Oder of operations)

In the case of ambiguity and is evaluated before or.

Example

```
True or False and False
 == True or (False and False)
 == True
```

Definition (Not)

The logical statement ‘ `not` ’ is the negation of logical statement:

$$\texttt{not True} == \texttt{False}\ ,$$

and

$$\texttt{not False} == \texttt{True}\ .$$

Question

Trace the following.

```
>>> a = 6
>>> b = 7
>>> a == 6
True
>>> a == 6 and b == 7
True
>>> not(a == 7 and b == 6)
True
>>> a != 7 and not b != 7
True
```

Short Circuits / Lazy Computation

```
>>> True or 1/0
True
>>> False or 1/0
ZeroDivisionError: division by zero
>>> True or laksdhalshd
```

`True` *Python does not bothering looking for* `laksdhalshd`

```
>>> True or !
  True or !
          ^
SyntaxError: invalid syntax
```

Short Circuits / Lazy Computation

```
>>> True and 1/0
ZeroDivisionError: division by zero

>>> False and 1/0
False
```

```
>>> a = 3
>>> a
3
>>> a == 3
True
>>> a != 3
False

>>> b = (a != 3)
>>> b
False
```

```
>>> True + 1
2

>>> 7*False
0

>>> True == 1
True

>>> False == 0
True
```

Contradiction

A contradiction is anything evaluating to `False` in all cases.

Example

The statements

1. `False` ,
2. `P and not P`.

are always `False` .

Tautologies

A tautology is anything evaluating to `True` in all cases.

Example

The statements

1. `True` ,
2. `P or not P`.

are always `True` .

Distribution of `not`

1. `not(a and b) == not a or not b`
2. `not(a or b) == not a and not b`

Example

`not(a and b or c)`

To eliminate ambiguity bracket the `and` *which gets evaluated first.*

```
== not( (a and b) or c)

== not(a and b) and not c

== not a or not b and not c
```

Question

Simplify `not(b or not a or not b).`

```
not(b or not a or not b)
 == not( (b or not a) or (not b) )
 == not(b or not a) and not(not b)
 == not b and not(not a) and b
 == not b and a and b
 == a and b and not b
 == a and False
 == False
```

Let us confirm with Python.

Question

Simplify

```
not( (a or b)) and not(b or not a or not b) )
```

```
not( (a or b) and not(b or not a or not b) )
 == not( (a or b) and False )
 == not(False)
 == True
```

Let us confirm with Python.

Question

Write a function

1. `is_even(x:int) -> bool` that returns `True` when an integer is even (and `False` otherwise);

2. `is_odd(x:int) -> bool` that returns `True` *only* when an integer is odd.

Answer

```
def is_even(x:int) -> bool:
    return x % 2 == 0


def is_odd(x:int) -> bool:
    return not is_even(x)
```

Question

Write a function

```
divides(x:int, y:int) -> bool
```

that returns `True` only when x divides y. That is to say, there is $t \in \mathbb{Z}$ such that $y = tx$.

Answer

```
def divides(x:int, y:int) -> bool:
    return y % x == 0
```

Next Time

1. More on functions.
2. Scope.