Booleans, Logical Expressions, and Predicates

CS111 Computer Programming
Department of Computer Science
Wellesley College

Making Decisions

If it's raining then bring umbrella and wear boots.
True or False?

Relational Operators

Booleans most naturally arise in the context of relational operators that compare two values.

```
In [1]: 3 < 5
Out[1]: True

In [2]: 3 < 2
Out[2]: False

In [3]: 3 > 2
Out[3]: True

In [4]: 5 <= 1
Out[4]: False

In [5]: 5 >= 1
Out[5]: True

In [6]: 5 == 5
Out[6]: True

In [7]: 5 == 6
Out[7]: False

In [8]: 5 != 6
Out[8]: True
```

Note == is pronounced "equals" and != is pronounced "not equals". This is why we distinguish the pronunciation of the single equal sign = as "gets", which is assignment and nothing to do with mathematical equality!

New values: Booleans

Python has two values of `bool` type, written `True` and `False`. These are called logical values or Boolean values, named after 19th century mathematician George Boole.

The values must be capitalized.

```
In [1]: True
Out[1]: True

In [2]: type(True)
Out[2]: bool

In [3]: true
NameError: name 'true' is not defined
```

Concepts in this slide:
Real-life examples for decision making with Boolean values.

New operators: relational. They are: >, <, >=, <=, ==, !=

“equals”

“not equals”
Relational Operators [cont.]

The relational operators can also be used to compare strings (in dictionary order):

```
In [1]: 'bat' < 'cat'
Out[1]: True

In [2]: 'bat' < 'ant'
Out[2]: False

In [3]: 'bat' == 'bat'
Out[3]: True

In [4]: 'bat' < 'bath'
Out[4]: True

In [5]: 'Cat' < 'bat'
Out[5]: True
```

In Python (and most other programming languages) uppercase letters come before lowercase letters in string ordering. See Digging Deeper section about the reason.

Logical Operators in plain English

- **a**: the cake has pineapple  False
- **b**: the cake is chocolate  True
- **c**: the cake has walnuts  True
- **d**: the cake is square  False

**Not**
- **not a**: the cake does not have pineapple  True/False?

**And**
- **a and b**: the cake has pineapple & the cake is chocolate  True/False?
- **b and c**: the cake is chocolate & the cake has walnuts  True/False?

**Or** (slightly different from English...)
- **a or b**: the cake has pineapple or the cake is chocolate  True/False?
- **b or c**: the cake has chocolate or the cake has walnuts  True/False?
- **a or d**: the cake has pineapple or the cake is square  True/False?

Logical Operators: not, and, or

**not exp evaluates to the opposite of the truth value of exp**

```
In [1]: not (3 > 5)
Out[1]: True

In [2]: not (3 == 3)
Out[2]: False
```

**exp1 and exp2 evaluates to True iff both exp1 and exp2 evaluate to True.**

```
In [3]: (3 < 5) and ('bat' < 'ant')
Out[3]: False

In [4]: (3 < 5) and ('bat' < 'cat')
Out[4]: True
```

**exp1 or exp2 evaluates to True iff at least one of exp1 or exp2 evaluates to True.**

```
In [5]: (3 > 5) or ('bat' < 'cat')
Out[5]: True

In [6]: (3 > 5) or ('bat' < 'ant')
Out[6]: False
```
Truth Tables: **and**

<table>
<thead>
<tr>
<th>exp1</th>
<th>exp2</th>
<th>exp1 and exp2</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

Truth Tables: **or**

<table>
<thead>
<tr>
<th>exp1</th>
<th>exp2</th>
<th>exp1 or exp2</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

**Concepts in this slide:**

and/or expressions produce different Boolean values.

---

**Predicates**

A **predicate** is simply any **function** that returns a Boolean value.

- **Determines if name is Darth Vader**
  ```python
def isDarth(name):
    return name == 'Darth Vader'
  ```

- **Determines whether num is divisible by factor**
  ```python
def isDivisibleBy(num, factor):
    return (num % factor) == 0
  ```

- **Determines whether n is even**
  ```python
def isEven(n):
    return isDivisibleBy(n, 2)
  ```

- **Determines whether strings s1 and s2 have the same length**
  ```python
def sameLength(s1, s2):
    return len(s1) == len(s2)
  ```

**Concepts in this slide:**

Definition and examples of predicates.

---

**More Predicates**

- **Determines if n is between lo and hi**
  ```python
def isBetween(n, lo, hi):
    return (lo <= n) and (n <= hi)
  ```

- **Determines if s is a Hogwarts House**
  ```python
def isHogwartsHouse(s):
    return (s == 'Gryffindor' or s == 'Hufflepuff' or s == 'Ravenclaw' or s == 'Slytherin')
  ```

- **Determines if n is a prime integer less than 100**
  ```python
def isSmallPrime(n):
    return (isinstance(n, int) and (n > 1) and (n < 100) and (n == 2 or n == 3 or n == 5 or n == 7 or not (isDivisibleBy(n,2) or isDivisibleBy(n,3) or isDivisibleBy(n,5) or isDivisibleBy(n,7))))
  ```

**Concepts in this slide:**

Examples of predicates with complex logical expressions.

---

**Preview: Some useful string operations**

We will cover strings and other “sequence” types like tuples and lists in a few lectures, but here are some useful operations that come handy when writing predicates.

The square bracket `[]` operator can be used to index (access) an element of a string.

- **In [1]:** `name = 'Esmeralda'
  ```
  In [2]: name[0]
  Out[2]: 'E'
  ```

  - **To notice:**
    - The index of the first character is 0 not 1, as you would expect. That is a quirk of many programming languages.
    - The method `lower` returns a new string that is the lowercased version of the original one, which doesn’t change. This behavior is different from `cs1graphics` objects.

- **In [3]:** `name[1]`

  - **Out[3]:** 'e'

- **In [4]:** `name.lower()`

  - **Out[4]:** 'esmeralda'

- **In [5]:** `name`

  - **Out[5]:** 'Esmeralda'

**Booleans**
Your Turn: Write these predicates

Exercise 1: Write the predicate `isVowel` that behaves as shown below:

```
In [6]: isVowel('E')
Out[6]: True
In [7]: isVowel('b')
Out[7]: False
```

Exercise 2: Use the predicate `isVowel` that you wrote above to write a new predicate `startsWithVowel` that behaves like shown:

```
In [8]: startsWithVowel('Esmeralda')
Out[8]: True
In [9]: startsWithVowel('bravery')
Out[9]: False
```

Short-circuit evaluation of `and` and `or`

In `exp1 and exp2` or `exp1 or exp2`, the expression `exp2` is not evaluated if the answer is determined by `exp1`.

```
In[14]: ((1/0) > 0) and (2 > 3)
ZeroDivisionError: integer division or modulo by zero
In[15]: (2 > 3) and ((1/0) > 0)
Out[15]: False
In[16]: (2 < 3) or ((1/0) > 0)
Out[16]: True
```

in and not in test for substrings

`s1 in s2` tests if string `s1` is a substring of string `s2`.

```
In [1]: 'i' in 'generation'
Out[1]: True
In [2]: 'u' in 'generation'
Out[2]: False
In [3]: 'era' in 'generation'
Out[3]: True
```

What other English words are in the string 'generation'?

```
In [6]: 'era' not in 'generation'
Out[6]: False
In [7]: 'get' not in 'generation'
Out[7]: True
```

Combining logical operators

What cake do I like?

```
(cake is chocolate) or (cake has pineapple) and (cake is square)
```

`and` takes precedence over `or` (like `*` over `+`)

```
((cake is chocolate) or (cake has pineapple)) and (cake is square)
```

Parentheses take precedence
**Long return expressions**

It's best to just use parens around long expressions. It is an unexpected but important Python fact that if you want to write long examples without the outermost parens on the return value, you must use the backslash continuation character to end the line (and this character cannot be followed by any other character except newline). Furthermore you must remove internal comments like `# Is n an integer?`

```python
    def isHogwartsHouse(s):
        return s == 'Gryffindor' or s == 'Hufflepuff' \\
            or s == 'Ravenclaw' or s == 'Slytherin'
```

**ASCII Table: uppercase vs. lowercase**

In computer programs, all data is stored as numbers (binary numbers made of 0 and 1s. Take CS 240 to learn more). ASCII is a standard that specifies the mapping between keyboard characters and numbers. When you compare “A” and “a”, you are comparing the underlying numbers 65 and 97.

**Simplifying logical expressions: Distributivity**

A: (cake has pineapple) and (cake has walnuts)

or (cake is chocolate) and (cake has walnuts)

B: (cake has pineapple or cake is chocolate) and (cake has walnuts)

**Simplifying logical expressions: De Morgan’s Laws**

Law 1.

(cake is not chocolate) and (cake has no walnuts)  
= not (cake is chocolate or cake has walnuts)

Law 2.

(cake is not chocolate) or (cake has no walnuts)  
= not (cake is chocolate and cake has walnuts)
**Truth Table for De Morgan’s first Law**

<table>
<thead>
<tr>
<th>chocolate</th>
<th>walnuts</th>
<th>not chocolate</th>
<th>no walnuts</th>
<th>not (chocolate and no walnuts)</th>
<th>chocolate or walnuts</th>
<th>not (chocolate or walnuts)</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>

Work out truth table for second law at home. 😊

**Test your knowledge**

1. What is the result of relational expressions? What is the result of logical expressions? What makes them different?
2. How does the comparison of string values work? Can you provide an example to illustrate?
3. Operators like >, or are called binary operators, while not is called a unary operator. Can you give an educated guess for the why?
4. [MATH] Relational operators are used in Math to describe intervals of numbers. Draw a picture showing the interval 10 to 20 (excluding 20). How would you write this in Python? What about the intervals of numbers less than 5 but greater than 15. Drawing the picture helps visualize relations.
5. Write the Truth Table for the expression not (exp1 and exp2)
6. Is there any difference between a predicate and a function?
7. Can you think of two predicates that one can write for the situations depicted in slide 6-2?
8. What is the result of the expression ‘$’ > ‘%’ . How would you explain that to someone?
9. In the expression 3 < 5 and ‘bat’ < ‘cat’ (notice there are no parens), does and have priority over <? Explain.