Type Boolean: bool

Lecture 02.02

By Marina Barsky

http://interactivepython.org/runestone/static/CS152f17/Selection/toctree.html

Logic in programming

- The Boolean (or logical) type is binary: it has only two values

True

False

```
male = True
old = False
```

Arithmetic relations

- Arithmetic relations often occur in logical conditions
- The relations compare two quantities of the same type (such as *int*s here):

  (a < b) which reads "a is less than b"

  (c > d) which reads "c is greater than d" or "c is more than d"

  (e <= f) which reads "e is less than or equal to f"

  (g >= h) which reads "g is greater than or equal to h"

  (i == j) which reads "i is equal to j"

  (j != k) which reads "j is not equal to k"

That is how we express equal!

All these relations produce True or False

Equivalent conditions

- Alternate or equivalent ways are possible to express the same condition:

p < q is equivalent to q > p

age < 12	⇔	12 > age
x <= y	⇔	y >= x
7 <= sum	⇔	sum >= 7
a > b	~~⇔~~	b<=a

Assignment of conditions to another variable

```
age = 24
over21 = (age > 21)
tied = (visitor_score == home_score)
error = (age < 0)
proper = (percent <= 100)
tall = (height >= 72) #inches
error2 = (denominator == 0)
```

Complements (opposites, negatives, inverses)

- Complements are logical opposites: when one is True the complement is False:

young vs. old

- Complements are expressed with the logical operator "not"
- The complement, or *not*, is *unary*: it acts on the one condition that follows it

b	**not b**
T	**F**
F	**T**

Truth table for NOT

Complements can involve arithmetic relations

(a < b) is the complement of (a >= b)

(a > b) is the complement of (a <= b)

(a == b) is the complement of (a != b)

young = not (age > 12);

- The above condition for *young* can be written without the not operator as:

not (age > 12) ⇔ (age <= 12)

not (age <= 21) ⇔ (age > 21)

Logical binary operators

- Operations on logical or Boolean boxes include two binary operators:
  - ***and*** - also called "andAlso"
  - ***or*** - also called "eitherOr“

- *Binary* operations (***or***, ***and***) operate on two operands: the operator is between the two **Boolean** operands

AND

- p and q is True when p is true **and** q is True

```
increasing = (x < y) and (y < z)
equilateral = (s1 == s2) and (s1 == s3)
is_in_range = (percent >= 0) and (percent <= 100)
is_eligible = over21 and is_employed
```

a	b	a and b
False	**False**	**False**
False	**True**	**False**
True	**False**	**False**
True	**True**	**True**

Truth table for AND

OR

- p or q is True when either p or q or both are True

```
win_point = (sum == 7) or (sum = 11)
error = (percent < 0) or (percent > 100)
play_ball = (inning <= 9) or (score1 == score2)
isosceles = (a == b) or (b == c) or (c == a)
```

a	**b**	**a or b**
False	**False**	**False**
False	**True**	**True**
True	**False**	**True**
True	**True**	**True**

Truth table for OR

From English to Python

- In English: play when the score is tied or time is not up and it's not raining.
- In Symbolic logic:

```
play_ball = ( (score1 == score2)
                        or (game_time < 90) )
                        and (not rain)
```

- Use parenthesis to ensure the order – *and* has a precedence over *or*

Illogic -- Looks good .. BUT is NOT

a and b < 7		(a < 7) and (b < 7)
a > b or c		(a > b) or (a > c)
a <= b and c	Should be ->	(a <= b) and (b <= c)
a == b == c		(a == b) and (b == c)
a == b and c		(a == b) and (a == c)
a != b or c		(a != b) and (a != c)
		not((a == b) or (a == c))

In Python

- All non-zero numbers are True
- All non-empty strings are True

Exercise 1

- The minimum passing grade is 50.
- Variable *grade* refers to the grade for a student. Select the expression(s) that correspond with the English sentence:

"The student passed."

A. grade >= 50

B. not (grade < 50)

C. 50 >= grade

D. not not (grade >= 50)

Exercise 2

- The minimum passing grade is 50. Consider this code:

```
>>> math_grade = 50
>>> history_grade = 85
```

- After the code above is executed, which expression(s) produce True?

A. history_grade == math_grade

B. (math_grade >= 50) and (history_grade >= 50)

C. (math_grade > 50) and (history_grade > 50)

D. (math_grade > 50) or (history_grade > 50)