Lecture 11: Booleans and conditionals

Chapter 7
CSCI 251 Introduction to Computer Science

Decision making in programs

- Python can execute different code depending on conditions.
- Statements that interrupt sequential flow of a program are known as **control structures**.
- **Decision structures** execute different code based on conditions.

Boolean Logic

- Named for British mathematician George Boole
  - developed symbolic logic at the heart of digital circuits
- Studied the "numbers" **true** and **false**, and the operations **or**, **and**, and **not**.
- The system he derived is called Boolean Algebra in his honor

Truth table

A	B	A and B	A or B	not A	not (A and B)	not (A or B)	not A or not B	not A and not B
True	**True**	True	True	False	False	False	False	False
True	**False**	False	True	False	True	False	True	False
False	**True**	False	True	True	True	False	True	False
False	**False**	False	False	True	True	True	True	True

The operators **and** as well as **or** can be used with more than two operands.
and is only True when all of its arguments are True.
or is True if any of its arguments are True.

For example, True **and** False **and** True evaluates to False
and, False **or** False **or** True **or** False evaluates to True

How about False or True or True and False (this is not clear, given what we've said so far)
how about True or (False and True)

Boolean type

- Boolean constants in Python are **`True`** and **`False`**
- These two values form a *type* in Python (called bool)
- It is the type of value calculated by **conditional expressions**

```
>>> done = False
>>> flag = True
>>> done or flag
True
>>> done and flag
False
>>> not flag
False
>>> not done
True
```

Conditional expressions

<expr> <relop> <expr>

- <expr> are (numerical) expressions
- <relop> is a **relational operator**.

Operator	**meaning**
<	less than
<=	less than or equal to
==	equal to
>=	greater than or equal to
>	greater than
!=	not equal to

Question: why do we use == instead of = for *equal to*?

Using relational operators

- Relational expressions allow us to compare two expressions (i.e. values) to each other

```
>>> x = 15
>>> x < 20
True
>>> y = 35
>>> y < 20
False
>>> y - x <= 20
True
```

Conditionals and the **`if`** statement

Simplest decision structure is the **`if`** statement

```
if <condition>:
    <body>
```

`<condition>` is an expression that is True or False

`<body>` is a sequence of indented Python statements

What does this function do?

```
def freezing():
    temp = float(input("Input temperature in
Celsius"))
    if (temp <= 0.0):
        print ("It's freezing!")
    if (temp > 0.0):
        print ("It's not freezing.")
```

Combining Boolean expressions

- Expressions can be combined using **and**, **or**, and **not**
  - result will also be a Boolean expression

True **and** True	_____	True **or** True	_____
True **and** False	_____	True **or** False	_____
False **and** True	_____	False **or** True	_____
False **and** False	_____	False **or** False	_____
not True	_____		
not False	_____		

Combining Boolean expressions

- Boolean expressions can be combined using **and**, **or**, and **not**

True **and** True	*True*	True **or** True	*True*
True **and** False	*False*	True **or** False	*True*
False **and** True	*False*	False **or** True	*True*
False **and** False	*False*	False **or** False	*False*
not True	*False*		
not False	*True*		

if with strings

- We can use **if** with **in** to test letters in a string

```
def justvowels(someWord):
    for letter in someWord:
        if letter in "aeiou":
            print (letter)

justvowels("hello there!")
```

Negative conditions using not

```
def notvowels(someWord):
    for letter in someWord:
        if letter not in "aeiou":
            print(letter)

notvowels("hello there!")
```

- What if our string is “Eat At Joe’s!"?

Dealing with case

```
def justvowels2(someWord):
    for letter in someWord:
        if letter in "aeiouAEIOU":
            print(letter)

def justvowels3(someWord):
    for letter in someWord:
        if letter.lower() in "aeiou":
            print(letter)
```

else

- When we use “if”, we can use “else” to handle the other cases

```
def wordInString(word, string):
  if word in string.split():
    print("Sure!“)
  else:
    print("No way!“)

>>> wordInString("class", "my favorite class is recess")
Sure!
>>> wordInString("lunch", "my favorite class is recess")
No way!
```

Even or odd

- We can get the remainder from dividing integers with the % operator. For example, 17%5 is 2 since 17 divided by 5 is 3 with a remainder of 2.
- What prints?

```
def  mod2( ):
   for index in range(10):
     print (index, "   ", index % 2)
```

Unzipping function

```
def separate(string):
  odds = ""
  evens = ""
  for index in range(len(string)):
    if index % 2 == 0:
      evens = evens + string[index]
    else:
      odds = odds + string[index]
  print ("Odds: ", odds)
  print ("Evens: ", evens)

>>> separate("rubber baby buggy bumpers")
Odds:  ubrbb ug upr
Evens:  rbe aybgybmes
```

While loops

- When we know how many times to loop, we use a for loop, which is a *definite* loop
  - for example, loop once for each letter in our word

- Other times, we don't know how many times to loop. In Python the most common indefinite loop is the **`while`** loop.

While loop syntax

```
while <condition>:
    body
```

```
def sayHiAgain():
 done = False
 while not done:
   print("hi")
   theirAnswer=input("Are you done?")
   if theirAnswer == "yes":
    done = True
```

```
>>> sayHiAgain()
hi
Are you done?no
hi
Are you done?no
hi
Are you done?no
hi
Are you done?yes
```

Infinite loops

- The statements in the **`while`** loop need to eventually make the Boolean expression false.

- Otherwise the loop will **never exit**
  - you have an infinite loop!