Lecture 4: Booleans and random

- Administrative
- for loops
- random module
- booleans
- conditionals
This week

- **Second assignment** due this coming Sunday.
- Command line interface,
- Drawing with the turtle module,
- Two short readings on AI + discrimination in hiring practices.
- Make sure you follow the [style guide](#) from now on.
Lecture 4: Booleans and random

- Administrative
- for loops
- random module
- booleans
- conditionals
Python for loops

- Python has a number of different "loop" structures that allow us to do repetition (computers are really good at doing repetitive tasks!)

- The `for` loop is one way of doing this

- There are a number of ways we can use the `for` loop, but for now the basic structure we'll use is:

```python
for some_variable in range(num_iterations):
    statement1
    statement2
    ...
```
Python for loops syntaxes

```
for some_variable in range(num_iterations):
    statement1
    statement2
    ...
```

- `for` is a keyword
- `in` is a keyword
- `range` is a function that we'll use to tell Python how many repetitions we want
- `num_iterations` is the number of iterations that we want the loop to do
- `some_variable` is a local variable whose scope (where it can be referred to) is only within the for loop
  - `some_variable` will take on the values from 0 to `num_iterations-1` as each iteration of the loop occurs
  - We're computer scientists so we start counting at zero :)
  - for example, in the first iteration, it will be 0, the second time 1, the third time 2, etc. we're computer scientists so we start counting at zero :)
- Don't forget the `:` at the end!
- Like with defining functions, Python uses indenting to tell which statements belong in the for loop
What would this code do?

```python
>>> for i in range(10):
...     print(i)
```

0
1
2
3
4
5
6
7
8
9
FOR LOOPS

What does this function do?

```python
>>> def sum(n):
    ...    total = 0
    ...
    ...
    ...    for val in range(n):
    ...        total = total + val
    ...
    ...    return total
```

- Sums and returns the numbers between 1 (well, 0) and n-1.
iterative_square function

```python
def iterative_square(length):
    for i in range(4):
        forward(length)
        right(90)
```

simple_star function

def simple_star():
    for i in range(36):
        forward(100)
        backward(100)
        right(10)
What if we wanted a star/asterisk with a different number of spokes?

```python
def asterisk_star(length, spokes):
    angle = \text{360} / \text{spokes}

    for i in range(spokes):
        forward(length)
        backward(length)
        right(angle)
```
simple_spiral function

```python
def simple_spiral():
    for i in range(50):
        forward(i * 5)
    right(55)
```
spiral function

def spiral(sides, angle):
    for i in range(sides):
        forward(i * 5)
        right(angle)
rotating_circles function

```python
def rotating_circles(radius, num):
    angle = 360 / num

    for i in range(num):
        circle(radius)
        right(angle)
```

FOR LOOPS
Lecture 4: Booleans and random

- Administrative
- for loops
- random module
- booleans
- conditionals
walk function

def walk(num_steps, step_size):
    for i in range(num_steps):
        angle = randint(-90, 90)
        right(angle)
        forward(step_size)
random module

- [http://docs.python.org/library/random.html](http://docs.python.org/library/random.html)
- It generates *pseudo-random* numbers
  - the numbers are not technically random, they're generated based on an algorithm (for most purposes, this is pretty good!)
- If you want truly random numbers, check out [http://www.random.org/](http://www.random.org/)
Useful functions

- `random` - returns a random float between 0 and 1.
- `uniform(a, b)` - returns a random float between $a$ and $b$.
- `randint(a, b)` - returns a random integer between $a$ and $b$.
- samples from many other distributions
  - `beta`
  - `exponential`
  - `gamma`
  - `normal`
Random Module

Importing only one function

- For now, we will only use the randint function.

- Rather than importing everything (*) we will be specific:

```python
from random import randint

>>> for i in range(100):
...     print(randint(0,10))
...
```

```python
8
9
5
0
1
7
```
**walk function**

```python
def walk(num_steps, step_size):
    for i in range(num_steps):
        angle = randint(-90, 90)
        right(angle)
        forward(step_size)
```
def pretty_picture():
    for i in range(10):
        # get some random values
        spokes = randint(5, 30)
        length = randint(10, 60)
        angle = randint(-90, 90)
        move = randint(20, 100)

        # move randomly somewhere else
        right(angle)
        forward(move)

        # draw a random star there
        asterisk_star(length, spokes)
def add_circles(number):
    """ Add number colored circles of radius 4 randomly through the screen """
    x_range = int(window_width() / 2)
    y_range = int(window_height() / 2)

    for i in range(number):
        x = randint(-x_range, x_range)
        y = randint(-y_range, y_range)

        # set the fill color of the circles
        # setcolor_xy(x, y)
        setcolor_random()

        pu()
        goto(x, y)
        pd()
        begin_fill()
        circle(8)
        end_fill()
Lecture 4: Booleans and random

- Administrative
- for loops
- random module
- booleans
- conditionals
Booleans

- So far, we have seen three types: `int`, `float`, `string`
- Python contains one more type, `bool` (stands for boolean)
- `bool` can only take the value True or False
- They generally result from asking T/F questions
T/F questions we can ask

- == (equal)
  - notice that '=' is the assignment operator while '==' asks whether two things are equal

- != (not equal)

- < (less than)

- > (greater than)

- <= (less than or equal to)

- >= (greater than or equal to)
Examples

```python
>>> 10 < 0
False
>>> 11 >= 11
True
>>> 11 > 11.0
False
>>> 11 > 10.9
True
>>> 10 == 10.1
False
>>> "test" == "test"
True
>>> "test" == "TEST"
False
>>> 10 != 10
False
>>> 10 != 11
True
>>> "banana" < "apple"
False
>>> type(True)
<class 'bool'>
>>> type(0 < 10)
<class 'bool'>
```
Combining booleans

- We can also combine boolean expressions to make more complicated expressions
- What kind of connectors might we want?
and

- `<bool expression> and <bool expression>`
- only returns True if both expressions are True
- otherwise, it returns False

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A and B</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

```python
>>> x = 5
>>> x < 10 and x > 0
True
>>> x = -1
>>> x < 10 and x > 0
False
```
or

- `<bool expression> or <bool expression>`
- returns True if either expression is True
- returns False only if both expressions are False

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A or B</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

```python
>>> x = 5
>>> x < 10 or x > 0
True
>>> x = -1
>>> x < 10 or x > 0
True
```
not

- not <bool expression>
- Negates the expression:
  - if the expression evaluates to True returns False
  - if the expression evaluates to False returns True

```
>>> not 5==5
False
```
Lecture 4: Booleans and random

- Administrative
- for loops
- random module
- booleans
- conditionals
if statement

- the key use of bool is to make decisions based on the answers
- the if statement allows us to control the flow of the program based on the result of a boolean expression
- if bool_expression:
  # do these statements if the bool_expression is True
  statement1
  statement2
  statement3
if statement

- the if statement is called a "control" statement in that it changes how the program flows

As the program runs, it evaluates the boolean expression. If it evaluates to True, it executes all of the statements under the if block and then continues on:

- It will execute statement1, statement2 and then statement3

- Otherwise, (i.e. the boolean expression evaluates to False), it will skip these statements and continue on (i.e. just execute statement3).
simple_if function

```python
def simple_if(num):
    """
    Given a number, prints out some comments based on the size of the number
    """
    if num > 10:
        print("That's a big number")
    print("I'm done")
```
input function

- Built-in function to read input from the keyboard
- It takes a string as a parameter and displays the string to the user
- Then waits for the user to enter some text. The program doesn't continue until the user hits enter/return
  - whatever the user typed will be returned by the input function as a string
- Note: if you want to convert the user input to a number, you need to use the `int(…)` or `float(…)` functions
If-else statement

- Sometimes we'd also like to do something if the bool expression evaluates to False. In this case, we can include an else statement.

- if <bool expression>:

  # execute these statements if the bool expression evaluates to True
  statement1
  statement2

else:

  # do these statements if the bool is False
  statement3
  statement4

statement5
If-else statement

- if the boolean expression evaluates to True,
  - execute statement1, statement2, then statement5
- else (i.e. the boolean expression evaluates to False)
  - execute statement3, statement4, then statement5.
def name_analysis():
    
    """
    Prompts the user for their name and gives a subjective analysis of the name
    """
    name = input("Enter your name: ")

    if name == "Alexandra" or name == "Zilong":
        print(name + " , that's a great name!")
    else:
        print(name + " , that name is ok!")

    print("Nice to meet you, " + name)
elif statement

- if <bool expression>:
  
  statement1

elif <bool expression>:

  statement2

  ... # we can have as many elif blocks as we want

else:

  statement3

statement4
elif statement

- The program starts with the first if statement.

- If it is True, it executes the statements in the if block (here, only statement1) then goes to the end (here, statement4) and continues

- If it is false, it goes to the first elif and checks if it is true. If it is true, it executes the statements in the elif block (here, statement2) then goes to the end (here, statement4) and continues

- The program will keep going down the list of elif statements as long as none of them are true

- If they are all false, then it will execute the statements under else

- elif avoids redundant calculations: if we know things are mutually exclusive, then once we find one that is true, we don’t check the others (jump directly outside the if-elif-else block)
**setcolor_xy function**

```python
def setcolor_xy(x, y):
    
    """ Set the fill color based on x, y coordinates """

    if x < 0 and y < 0:
        fillcolor("blue")
    elif x < 0 and y > 0:
        fillcolor("purple")
    elif x > 0 and y < 0:
        fillcolor("red")
    else:
        fillcolor("yellow")
```
setcolor_random function

def setcolor_random():
    """ Set the fill color randomly from: blue, purple, red and yellow """
    color = randint(1, 4)

    if color == 1:
        fillcolor("blue")
    elif color == 2:
        fillcolor("purple")
    elif color == 3:
        fillcolor("red")
    else:
        # color == 4
        fillcolor("yellow")
def temperature_report(temperature):
    
    """ Converts a numerical temperature to one of: hot, warm, cool or cold """

    if temperature > 80:
        temp = "hot"
    elif temperature > 70:
        temp = "warm"
    elif temperature > 50:
        temp = "cool"
    else:
        temp = "cold"

    return temp
Resources

- Textbook: Chapter 7 and Chapter 8.
- turtle-examples.txt
- conditional-turtle.txt
- conditionals.txt

Practice Problems

- Practice 2 (solution)

Homework

- Assignment 2