01-30-2023

CS051A

INTRO TO COMPUTER SCIENCE WITH TOPICS IN AI

4: Booleans and random

Alexandra Papoutsaki

she/her/hers

Lectures

Zilong Ye

he/him/his

Labs

Welcome to lecture 4, everyone! Are there any questions? How did the submission of the first assignment go?!

Lecture 4: Booleans and random

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

Let's start with our usual announcements and then we will proceed with Python.

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.

This week we will start working towards our second assignment which will introduce a new (well really old) way of interacting with our computers, the command line interface. The main deliverable of this assignment will be a picture that you will draw (and the code behind) using the turtle module. We will also have two short readings on AI and discrimination in hiring practices. Prof Ye asked me to share a style guide that you should follow when writing code for this class (and in general).

Lecture 4: Booleans and random

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

If there are no questions about logistics, let's get started with Python. We will resume where we left off with for loops.

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:

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

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:

```
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

Look at the syntax:
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?

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

```
0
1
2
3
4
5
6
7
8
9
```

What if I type:

```
for i in range(10):
        print(i)
```

This code will print the numbers 0 to 9 (remember we start counting at 0 and we stop one step before the number we passed to the range function.

What does this function do?

```
>>> 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.

Take a minute, what do you think that this function does?

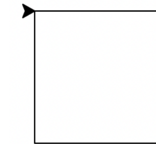
```
def sum(n):
    total = 0
    for val in range(n):
        total = total + val
    return total
```

It's a function with one parameter, a number n. It sums and returns the numbers between 1 (well, 0, but that doesn't count) and n-1.

iterative_square function

turtle-examples.py

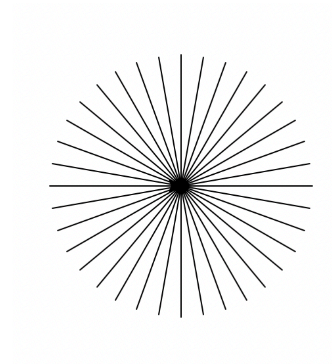
```
16  def iterative_square(length):
17      for i in range(4):
18          forward(length)
19          right(90)
20
```



Back to the turtle module. How can we use a for loop to draw a square? Take a look at the iterative_square method in the turte_examples.py file. We have bundled together the two statements in a for loop that is repeated 4 times.

simple_star function

```
def simple_star():
    for i in range(36):
        forward(100)
        backward(100)
        right(10)
```

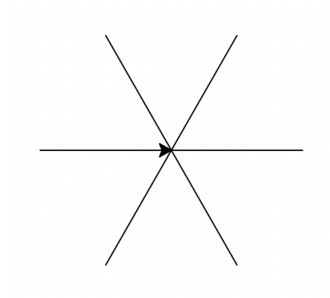


What about the simple_star function? It draws a 36-sided star (or asterisk)

What if we wanted a star/asterisk with a different number of spokes?

```
def asterisk_star(length, spokes):
    angle = 360 / spokes

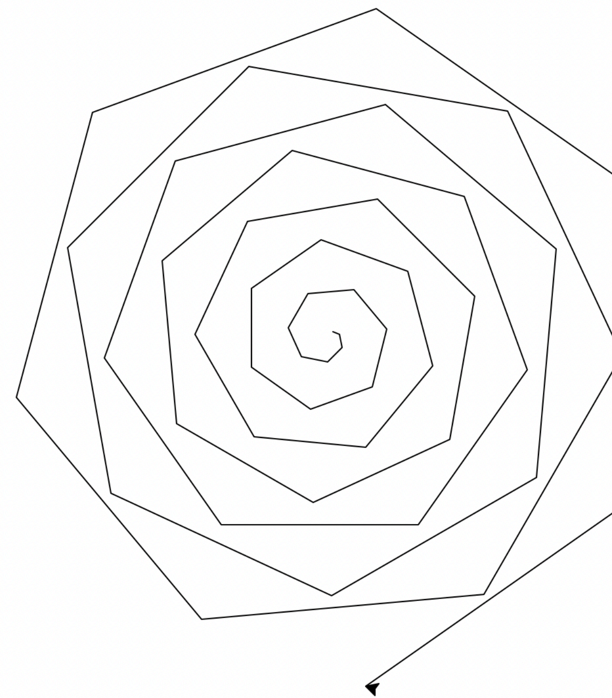
    for i in range(spokes):
        forward(length)
        backward(length)
        right(angle)
```



What if we wanted a star/asterisk with a different number of spokes? Look at the asterisk_star function. We first figure out how we have to space the spokes. We do a for loop over the number of spokes. At each iteration we draw a spoke, go backwards for the next spoke, rotate right based on the angle we calculated. Here's the result of asterisk_star(100, 6)

`simple_spiral` function

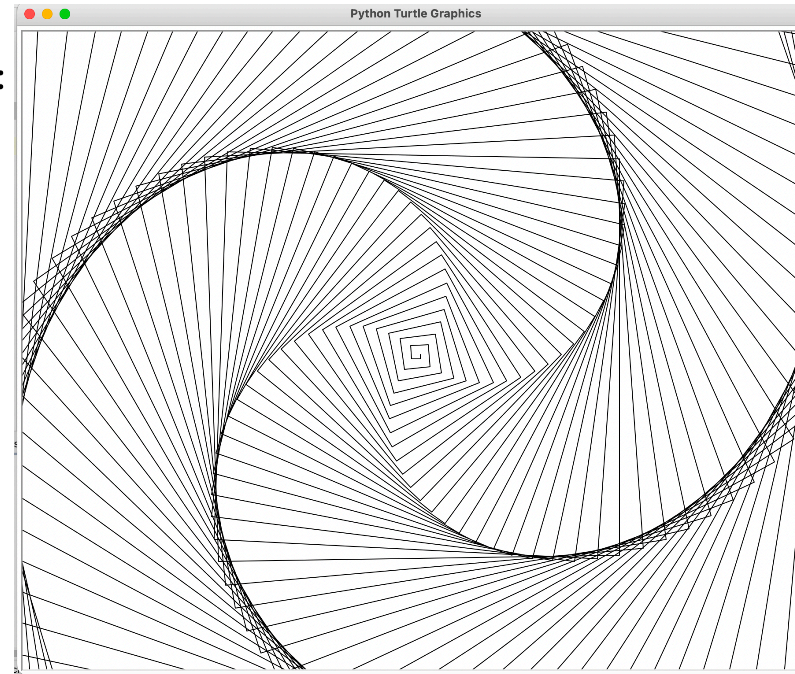
```
def simple_spiral():
    for i in range(50):
        forward(i * 5)
        right(55)
```



What about the simple_spiral function? It draws a… simple spiral (good naming helps)! Look at the code: The variable i will keep count of how many times we need to get in the loop. For small loops (i.e. just a handful of statements), it's common to just name the counter variable as i which stands for index. Each time we get in the loop, the length of the edge drawn will be longer by 5: 0, 5, 10, 15, 20, … and it will be at a 50 degree angle. If it is less than 90 degrees, it will spiral out, and above it, it will spiral in.

`spiral` function

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

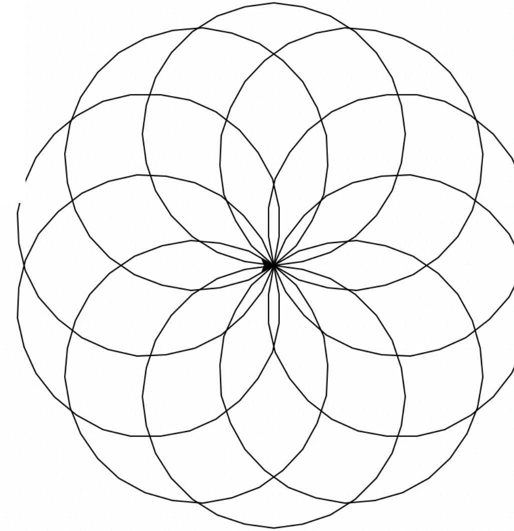


The spiral function is similar to simple_spiral, however now we've parameterized the length of the sides and the angle. A good example, is side = 200 and angle = 89

rotating_circles function

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

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



Finally, rotating_circles is another fun function that draws “num” circles, each one rotated "angle" degrees from the previous one. This is the result of the rotating_circles(100, 10) call.

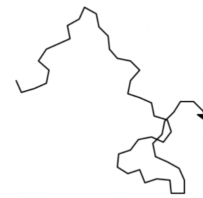
Lecture 4: Booleans and random

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

Now let's look into the walk function in turtle-examples.py

walk function

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



How is this accomplished? It turns a random angle between -90 and 90 and steps forward some step size for num_steps.

random module

- 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/

So far we have seen two modules, math and turtle. There is one more that will come handy: the random module. The random module contains functions that generate pseudo-random numbers. Why pseudo (==not genuine)? The numbers are not technically random, they are generated by an algorithm but for most purposes, including ours, they are random enough. If you want truly random numbers check out the website 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

Some functions that you might find useful are random (returns a number between [0,1), uniform (returns a float between [a,b] or [a,b) depending on rounding), and randint (returns an int between [a,b]), as well as samples from well-known distributions.

Importing only one function

- For now, we will only use the `randint` function.
- Rather than importing everything (*) we will be specific:

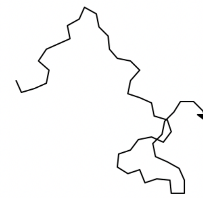
```
from random import randint
```

```
>>> for i in range(100):
...     print(randint(0,10))
...
8
9
5
0
1
7
```

For now, we are only interested in the randint function. Instead of loading all functions, we can be specific: from random import randint. This for loop prints 100 random integers between 0 and 10 (inclusive).

walk function

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



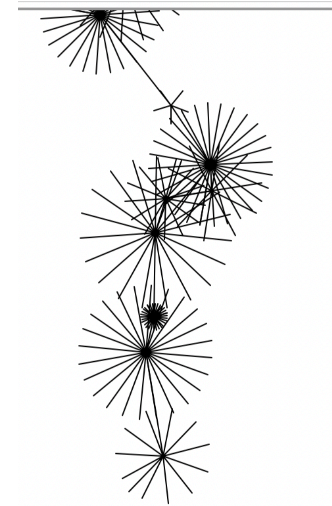
Now you understand better how this random walk was created.

`pretty_picture` function

```
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)
```



What does the pretty_picture function do? It draws a line of length between 10 and 60 at an angle between -90 and 90 then draws a star of length between 10 and 60 and with “spokes” spokes. It does that 10 times. Pretty, right?

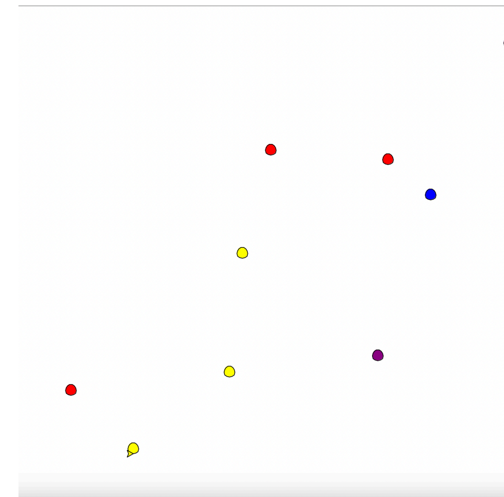
add_circles function

```
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()
```



Now let's look at the add_circles function in the conditional-turtle.py. It picks random x and y coordinates to draw a circle. It uses the randint function which we have seen before. Now how are the colors chosen? Each quadrant of the xy-axes is a different color. Wait, how can we do this? We will have to ask a question about x and y.

Lecture 4: Booleans and random

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

Which brings us to a new type that of booleans.

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

So far we have seen ints, floats, and strings. The fourth type we will encounter, bool is short for boolean, and it represents the answer to true/false questions (True or False in Python, mind the capitalization)

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)

Here are some examples of true/false questions we can ask on data we have seen so far. We can check equality (with the == operator, not = which is for assignment!), inequality, what is greater/greater or equal, less, or less than or equal to.

Examples

```
>>> 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'>
```

Here are a few boolean expressions and what they evaluate to. You can see we can work not only with numbers but strings too.

Combining booleans

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

The nice thing is that we can combine booleans to build more complex logical questions. What kind of ways can we connect boolean expressions?

and

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

A	B	A and B
T	T	T
T	F	F
F	T	F
F	F	F

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

The first one will be using the and operator that evaluates two expressions and returns true only if both are true.

or

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

A	B	A or B
T	T	T
T	F	T
F	T	T
F	F	F

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

The or operator will return true as long as at least one of the two expressions evaluates to true.

not

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

A	not A
T	F
F	T

```
>>> not 5==5
False
```

The not expression negates the expression. This one can be a bit confusing so make sure you first evaluate the expression (e.g., 5==5 is True) and then negate it (turns it into False)

Lecture 4: Booleans and random

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

Where can we use booleans?

`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
  ```

The whole point of booleans is to make decisions based on the T/F answer they give us. We will use the if statement to control the flow of the program based on the result of a boolean expression. Its syntax is 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`).

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

```
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")
```

Look at the simple_if function in conditionals.py. It takes one number and prints out “That’s a big number” if the number is 10. Regardless of what the number is, it will 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

Let's see how we can ask the user for input. We will use the built-in function input which takes a string as a parameter and displays the string to the user once they hit enter. Please note that even if the user enters a number, the number will be read as a string. You would need to convert (cast it) to its numerical representation using the int or float function.

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
  ```

Sometimes, we would 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.`

What does the syntax mean? if the boolean expression evaluates to True, then the interpreter executesstatement1, statement2, then statement5. else (i.e. the boolean expression evaluates to False) it will execute statement3, statement4, then statement5.

name_analysis function

```
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)
```

Let’s look at the function name_analysis in conditionals.py. We first use the input function to get the user's name. input returns the text the user entered which we store in the variable “name”.
name == “Alexandra" or name ==“Zilong” checks whether the entered name is “Alexandra" or “Zilong”. The if statement directs the program's behavior depending on the answer. Finally, regardless of the name, we print out "Nice to meet you…" (let’s pretend we’re polite after telling the non-Alexandras and Zilongs of the world that their name is just ok).

`elif` statement

- ```
  if <bool expression>:
      statement1
  elif <bool expression>:
      statement2
      ... # we can have as many elif blocks as we want
  else:
      statement3

  statement4
  ```

Sometimes, there are multiple boolean expressions we want to check if they are true. In this case, we will use the elif statement which has the following syntax:

```
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)

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.

Why would we want to use elif? 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 (we just jump directly outside the if-elif-else block)

setcolor_xy function

```
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")
```

Looking back at the turtle-conditional.py and specifically the setcolor_xy function, we notice that it uses the if-elif-else statement to select between the four options.

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")
```

What about the setcolor_random function? It randomly picks between blue, purple, red and yellow (instead of based on x, y). Any ideas on how could we get this behavior? We will use random.randint to select a number between 1 and 4. We will save this number and use it in an if-elif-else statement. We MUST save this number to a variable and not try and do your if/else statement based on new calls to random.randint!!!

temperature function

```
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
```

Let’s now take a look at the temperature function in conditionals.py. Essentially it takes a number and does the following checks:
- > 80 => "hot"
- 71 - 80 => "warm"
- 51 - 70 => "cool"
- <= 50 => “cold"

Make sure you check the rest of the forecast functions in the file.

Resources

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

Practice Problems

- Practice 2 (solution)

Homework

- Assignment 2