Learning Goals

- Practice conditionals and \texttt{for} loops
- Use Turtle Graphics
Reading and Lab

• This week: 1.6 – 1.11, Chapter 2
• Lab this week…
  – Please check in at 3 PM to get credit for lab
    • stay until done with lab problems or 5 PM, whichever comes first ;^)
  – Practice with for loops and if-elif-else
  – Getting ready for gene finding!
  – Some fun optional bonus problems

The reading reinforces the lecture material.
Gauging your Workload

On your worksheet…

   On the front, bottom-right corner…

How many hours, outside of class or lab time, did you spend on this course this past week?
def perfect(n):
    """Returns True is n is perfect, False otherwise""
    sod = 0
    for d in range(1, n):
        if n % d == 0:
            sod = sod + d
    return n == sod

def collect(up_to):
    """What does this do?""
    perfects = []
    for i in range(1, up_to):
        if perfect(i):
            perfects = perfects + [i]
    return perfects

We tried to run this function with
up_to = 100000...
Approximating pi!

Use “darts” to approximate pi!
- generate darts (random numbers inside unit square)
- count number of darts that land in unit circle
- \( \pi = 4 \times \frac{\text{“inside darts”}}{\text{“total darts”}} \)

```python
import random

def estimate_pi(num_darts):
    # YOUR CODE HERE
```

The random module has a function `uniform(a, b)` that returns a random number in the range [a, b].
def mystery1(n):
    for k in range(1, n):
        if k*k == n: return True
    return False

def mystery2(n):
    for k in range(1, n):
        if n == 1:
            return True
        elif not n % 2 == 0:  # n % 2 != 0
            return False
        else:
            n = n/2

Assume that we run these functions only with positive integers as input!

Try these on your worksheet!
Collatz Revisited

```python
def collatz(n):
    """Returns n/2 if n is even and returns 3n+1 otherwise""
    if n % 2 == 0:   # if n is even...
        return n/2
    else:
        return 3*n + 1

>>> test_num(16, 10)
True
```

If we start with 16 and apply collatz repeatedly, do we get to 1 within the first 10 repeats?

```python
def test_num(number, repeats):
    """Returns True if the number collatzes within the given number of repeats""
    for i in range(repeats):
        number = collatz(number)
        if number == 1: return True
    return False
```
Collatz Re-Revisited

```python
def test_num(number, repeats):
    """Returns True if the number collatzes within the given number of repeats""

for number in range(2, up_to+1):
    Try all numbers from 1 to 20 each time!

Fill in the missing parts!
```
Collatz Re-Revisited

```python
def test_conjecture(up_to, repeats):
    """Determines if all numbers from 1 to up_to collatz to 1 within given number of repeats""
    for number in range(2, up_to+1):
        if not test_num(number, repeats):
            return False
    return True

def test_num(number, repeats):
    """Returns True if the number collatzes within the given number of repeats""
```

```python
>>> test_conjecture(20, 10)
False
>>> test_conjecture(20, 50)
True
```

Try all numbers from 1 to 20 Up to this many repeats each time!

Demo `collatz.py`
From xkcd...

The Collatz Conjecture states that if you pick a number, and if it's even divide it by two and if it's odd multiply it by three and add one, and you repeat this procedure long enough, eventually your friends will stop calling to see if you want to hang out.
def leppard(input_string):
    """What does this do?""
    output_string = ""
    for symbol in input_string:
        if symbol == "o":
            output_string = output_string + "ooo"
        else:
            output_string = output_string + symbol
    return output_string

>>> leppard("hello")

>>> leppard("hello to you")
def leppard(input_string):
    """What does this do?"""
    output_string = ""
    for symbol in input_string:
        if symbol == "o":
            output_string = output_string + "ooo"
        else:
            output_string = output_string + symbol
    return output_string

>>> leppard("hello")
"hellooroo"

>>> leppard("hello to you")
"hellooroo tooo yooou"
This is not part of Python (“home brewed” by Prof Wu)

```python
import speech

def leppard(input_string):
    """Leppard-ifies the input string and speaks it!""
    output_string = ""
    for symbol in input_string:
        if symbol == "o":
            output_string = output_string + "ooo"
        else:
            output_string = output_string + symbol
    return output_string

speech.say("hello")
speech.say(leppard("hello"))

speech.say("hello to you")
speech.say(leppard("hello to you"))
```

Demo leppard.py
z detector

```python
def z(input):
    counter = 0
    for symbol in input:
        if symbol == 'z':
            counter = counter + 1
    return counter
```

```python
def z(input):
    counter = 0
    for i in range(len(input)):
        if input[i] == 'z':
            counter = counter + 1
    return counter
```

The “direct” method

The “indirect” or “index” method
Spam counter!

```python
def spam_count(input):
    counter = 0
    for letter in input:
        ???

>>> spam_count("I like spam with spamspamspam!")
4
>>> spam_count("spamityspampampam!")
2
```

First attempt…
The direct method
Spam counter!

```python
def spam_count(input):
    counter = 0
    for i in range(len(input)):
        # Finish this in your notes. Hint: Use slicing! (e.g., input[2:5])
    Ah, the indirect/index method!
```

```
>>> spam_count("I like spam with spamspamspam!")
4
012345678911111111
01234567

>>> spam_count("spamityspampampam!")
2
```
Spam counter!

```python
def spam_count(input):
    counter = 0
    for i in range(len(input)):
        if input[i:i+4] == "spam":
            counter = counter + 1
    return counter
```

Notice that there is no need for an else after the if!
Functions that return lists

```python
def squares(n):
    output = []
    for x in range(1, n+1):
        output = output + [x*x]  # upgrade to list-hood!
    return output
```

```python
def squares(n):
    output = []
    for x in range(1, n+1):
        output.append(x*x)
    return output
```

```python
>>> squares(5)
[1, 4, 9, 16, 25]
```
Spam finder!

01234567891

>>> spam_finder("spampamity")
[0, 4]

>>> spam_finder("ssspam!")
[2]

def spam_finder(input):

Spam finder!

```python
def spam_finder(input):
    output = []
    for i in range(0, len(input)):
        if input[i:i+4] == "spam":
            output.append(i)
    return output
```

```python
>>> spam_finder("spamspamity")
[0, 4]
>>> spam_finder("ssspam!")
[2]
```
Stepping!

def return_codons(DNA_string):
    codon_list = []
    for i in range(0, len(DNA_string), 3):
        codon_list.append(DNA_string[i:i+3])
    return codon_list

>>> return_codons("AAATTTGGGC")
["AAA", "TTT", "GGG", "C"]
Turtle Graphics

Logo (programming language) [1967]
Meet Python’s Turtle…

```python
>>> import turtle

>>> turtle.forward(100)

>>> turtle.right(90)
```

Hey, is it legal to import turtles?

DEMO!
import turtle

def square(length):
    """Draws a square with given side length"""
    for x in range(0, 4):
        turtle.forward(length)
        turtle.right(90)

def polygon(length, sides):
    """Draws a polygon with given side length and number of sides"""
    for x in range(0, sides):
        turtle.forward(length)
        turtle.right(360.0/sides)

Notice that this for loop is just being used to repeat something 4 times!

Also nothing is returned by these functions!

DEMO!
import turtle

def polygon(length, sides):
    for x in range(0, sides):
        turtle.forward(length)
        turtle.right(360.0/sides)

def spirograph(length, sides, polys):
    for iteration in range(0, polys):
        polygon(length, sides)
        turtle.right(360.0/polys)

>>> spirograph(50, 6, 10)  

DEMO!
Approximating pi!

Use “darts” to approximate pi!
• generate darts (random numbers inside unit square)
• count number of darts that land in unit circle
• $\pi = 4 \times \text{“inside darts”} / \text{“total darts”}$

```python
import random

def estimate_pi(num_darts):
    # YOUR CODE HERE
```

The random module has a function `uniform(a, b)` that returns a random number in the range $[a, b]$. 
def estimate_pi(num_darts):
    # number of "inside" darts
    num_inside = 0

    for i in range(num_darts):
        # sample x- and y-coordinates uniformly in unit square
        x = random.uniform(-1, 1)
        y = random.uniform(-1, 1)

        # compute distance to origin
        dist = (x**2 + y**2) ** 0.5

        # inside unit circle?
        if dist < 1:
            num_inside = num_inside + 1

    # approximate pi
    return 4 * num_inside / num_darts