Learning Goals

• Practice conditionals and 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
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 if 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...
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
```
More Mysteries!

```python
def mystery1(n):
    for k in range(1, n):
        if k**2 == n:
            return True
    return False

def mystery2(n):
    for k in range(1, n):
        if n == 1:
            return True
        elif not 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:
        return n/2  # if n is even...
    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?

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
```

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

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

def test_conjecture(up_to, repeats):
    """Determines if all numbers from 2 to up_to collatz to 1 within given number of repeats""
    for number in range(2, up_to+1):
        # Fill in the missing parts!
```

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

Try all numbers from 1 to 20 Up to this many repeats each time!
Collatz Re-Revisited

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

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

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

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

demo collatz.py
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")
Mystery

I love a good mystery!

```python
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")
"hellooo"

>>> leppard("hello to you")
"hellooo 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"))
```
z detector

>>> z("I like zyzzyvas!")
3
>>> z("I am opposed to the letter after y")
0

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

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

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

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

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

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
```

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

```python
def spam_finder(input):
    # Implementation of spam finder algorithm
    # Example usage:
    >>> spam_finder("spamspamity")
    [0, 4]
    >>> spam_finder("ssspam!")
    [2]
```

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

What colorful codons you have!
Turtle Graphics

Logo (programming language) [1967]

forward 50  right 90  forward 50  right 90

forward 50  right 90  forward 50  right 90
>>> import turtle

>>> turtle.forward(100)

>>> turtle.right(90)
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)
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
- \( \pi = 4 \times \frac{\text{inside darts}}{\text{total darts}} \)

```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 \([a, b]\).
Approximating pi!

def estimate_pi(num_darts):
    num_inside = 0

    for i in range(num_darts):
        x = random.uniform(-1, 1)
        y = random.uniform(-1, 1)

        dist = (x**2 + y**2) ** 0.5

        if dist < 1:
            num_inside = num_inside + 1

    return 4 * num_inside / num_darts