CSCE 110 — Programming I
Simulations and Plotting Data

Dr. Tiffani L. Williams

Department of Computer Science and Engineering
Texas A&M University

Fall 2011
We are now done with Part 1 of the course.

- We have covered the basics of Python.
- Now, it’s time to look at different kinds of problems we can solve using our new found knowledge of Python.
- In this module, we will look at simulation and plotting data.
Flipping a coin

- On average, how often will heads appear during a coin flip?
- How would you prove it?
# Fair coin experiments

**Listing 1: fair-coin.py**

```python
# A program that simulates the number of heads and tails that appear
# using a fair coin.

import random

def flip_coin():
    return random.choice(['heads', 'tails'])

def main(coin_tosses):
    heads = 0
tails = 0
    for i in range(0, coin_tosses):
        result = flip_coin()
        if result == 'heads':
            heads += 1
        else:
            tails += 1

    # Print results
    print 'Total number of coin tosses: %d' % (coin_tosses)
    print 'Number of heads: %d (%.2f percent)' % (heads, float(heads)/coin_tosses * 100)
    print 'Number of tails: %d (%.2f percent)' % (tails, float(tails)/coin_tosses * 100)

main(10000)
```
Unfair coin experiments

Listing 2: unfair-coin.py

```python
# A program that simulates the number of heads and tails that appear
# using an unfair coin.

import random

def flip_coin():
    return random.choice(['heads', 'tails', 'tails', 'tails'])

def main(coin_tosses):
    heads = 0
    tails = 0
    for i in range(0, coin_tosses):
        result = flip_coin()
        if result == 'heads':
            heads += 1
        else:
            tails += 1

    # Print results
    print 'Total number of coin tosses: %d' % (coin_tosses)
    print 'Number of heads: %d (%.2f percent)' % (heads, float(heads)/coin_tosses * 100)
    print 'Number of tails: %d (%.2f percent)' % (tails, float(tails)/coin_tosses * 100)

main(10000)
```
Consider the following problem.

On average, how many times do you need to roll a die before all six different numbers have turned up?
We can use a computer to simulate the problem for us.

**Listing 3: rolling-all-numbers.py**

```python
import random

def roll_die():
    return random.randint(1,6)

def roll_all_numbers():
    rolled_value = [0,0,0,0,0,0,0]
    number_rolls = 0

    while sum(rolled_value) != 6:
        number = roll_die()
        rolled_value[number] = 1
        number_rolls += 1

    return number_rolls

def conduct_simulation(repetitions):
    total_rolls = 0
    for i in range(0, repetitions):
        total_rolls += roll_all_numbers()
    print 'Repetitions: %-8d 	 Average rolls is %.1f' % (repetitions, float(total_rolls) / repetitions)

def main():
    for experiment in (1, 10, 100, 1000, 10000, 100000):
        conduct_simulation(experiment)

main()
```
Consider the following problem.

Let’s assume you roll six dice at the same time. On average, how many times do you need to roll six dice before each of them has a different number?
Computer programming comes to the rescue again.

Listing 4: six-dice-with-all-numbers.py

```python
# Simulates the number of times we have to roll six dice so that each of them
# have different values.
import random

def roll_six_dice():
    roll = []
    for die in range(6):
        roll += [random.randint(1, 6)]
    return roll

def roll_all_numbers():
    number_rolls = 0
    roll = []
    while len(roll) != 6:
        roll = set(roll_six_dice())
        number_rolls += 1
    return number_rolls

def conduct_simulation(repetitions):
    total_rolls = 0
    for i in range(0, repetitions):
        total_rolls += roll_all_numbers()
    print 'Repetitions: %-8d  \t Average rolls is %f' % (repetitions, float(total_rolls) / repetitions)

def main():
    for experiment in (1, 10, 100, 1000, 10000):
        conduct_simulation(experiment)

main()
```
Sets

- A set is used to contain an unordered collection of objects. The elements of a set are never duplicated.
- To create a set, use the `set()` function and supply a sequence of items such as follows:
- Sets are unordered and cannot be indexed by numbers.
- Sets support a standard collection of operations, including union, intersection, difference, and symmetric difference.
- New items can be added to a set using `add()` or `update()`.
Set Examples (1)

>>> a = [1, 2, 3, 4, 1, 3, "apple"]
>>> b = set(a)  # convert the list a to a set
>>> b
set([1, 2, 3, 4, 'apple'])
>>> c = set(['apple', 'banana', 3])
>>> c
set([3, 'apple', 'banana'])
>>> c.add(4)  # add the integer 4 to the set c
>>> c
set([3, 4, 'apple', 'banana'])
>>> c.add("banana")  # add "banana" to the set c; but "banana" is already there
>>> c
set([3, 4, 'apple', 'banana'])
>>> b.union(c)  # union
set([1, 2, 3, 4, 'apple', 'banana'])
>>> b.intersection(c)  # intersection
set([3, 4, 'apple'])
>>> b.difference(c)  # set difference
set([1, 2])
>>> b.issubset(c)  # subset
False
>>> b.issuperset(c)  # superset
False
>>> d = set([1,2])
>>> d.issubset(b)  # subset
True
Set Examples (2)

```python
>>> b
set([1, 2, 3, 4, 'apple'])
>>> b.pop()  # arbitrarily select an item from the set b
1
>>> b
set([2, 3, 4, 'apple'])
>>> b.remove("apple")  # remove "apple" from the set b
>>> b
set([2, 3, 4])
>>> len(b)
3
>>> b[1]
Traceback (most recent call last):
  File "<string>"., line 1, in <fragment>
TypeError: 'set' object does not support indexing
```