Prediction is difficult, especially about the future …

```python
# File: chaos.py
# A simple program illustrating chaotic behavior
# ** this does no error checking **
# Matt Bishop, ECS 10, Spring 2014
#
# announce what the program does
print("This program illustrates a chaotic function")

# ask, and convert it to a float
x = float(input("Enter a number between 0 and 1: "))

# now apply the chaotic function 100 times
for i in range(100):
    x = 3.9 * x * (1 - x)
    print(x)
```

```math
x = x + 1
CS \
increment x
```
Sample Run …

```
>>> ============ RESTART ============
>>> This program illustrates a chaotic function
Enter a number between 0 and 1: 0.25
0.73125
0.76644140625
0.6981350104385375
0.8218958187902304
0.5708940191969317
0.9553987483642099
0.166186721954413
0.5404179120617926
0.9686289302998042
0.11850901017563877
0.4074120362630336
0.9415671289870646
0.214572035332672
0.6572704202448796
0.8785374581723959
0.4161666317654883
0.9475906688447814
0.1936841133601687
0.6090652525513056
0.9286086056750876
0.25854918625090323
0.747635867705606
0.7358382604001973
0.7580832282324941
0.7152328844898681
0.7943317411932672
0.6371384218919443
0.9016529076398497
0.3458322729593719
0.8823060165625929
0.4049842278301656
0.9397908118519834
0.2206777630612359

# File: chaos.py
# A simple program illustrating chaotic behavior
#
# ** this does no error checking **
# Matt Bishop, ECS 10, Spring 2014
#
# announce what the program does
print("This program illustrates a chaotic function")
#
# ask, and convert it to a float
x = float(input("Enter a number between 0 and 1: "))
#
# now apply the chaotic function 100 times
#
for i in range(100):
x = 3.9 * x * (1 - x)
print(x)
```
Where chaos reigns ...
# File: chaos.py
# A simple program illustrating chaotic behavior
#
# ** this does no error checking **
#
# Matt Bishop, ECS 10, Spring 2014
#
# announce what the program does
print("This program illustrates a chaotic function")
#
# ask, and convert it to a float
x = float(input("Enter a number between 0 and 1: "))
#
# now apply the chaotic function 100 times
#
for i in range(100):
    x = 3.9 * x * (1 - x)
    print(x)
# File: chaos.py
# A simple program illustrating chaotic behavior
#
# ** this does no error checking **
#
# Matt Bishop, ECS 10, Spring 2014
#
# announce what the program does
print("This program illustrates a chaotic function")
#
# ask, and convert it to a float
x = float(input("Enter a number between 0 and 1: "))
#
# now apply the chaotic function 100 times
#
for i in range(100):
    x = 3.9 * x * (1 - x)
    print(x)

ValueError: could not convert string to float: '0.3+0.5'

Program output:
This program illustrates a chaotic function
Enter a number between 0 and 1: 0.3+0.5
Lots of rice: doubling from one square to the next … … …
Exception Handling

```python
# File: divby0.py
# See what happens when you divide by 0
# #
# # Note: to get to the last two parts (where this catches the exception),
# # comment out the first division
# #
# Matt Bishop, ECS 10, Spring 2014
#
#
# this can be any integer
# -- if you want to see the difference between the two catches,
# set this to a string like 'hello'
x = 7
#
# Divide, and bomb
y = x / 0
#
# Here's how you check for it
try:
    y = x / 0
except:
    print("Error occurred")
#
# And here you catch *only* the division by zero
try:
    y = x / 0
except ZeroDivisionError:
    print("You can't divide by 0!")
    print("Really, you can't!")
except TypeError:
    print("You can't divide these types!")```
Traceback (most recent call last):
  File "/Users/ludaesch/Dropbox/10-SQ-2014/programs/divby0.py", line 18, in <module>
    y = x / 0
ZeroDivisionError: division by zero

Error occurred
You can't divide by 0!
Really, you can't!

# Divide, and bomb
y = x / 0

# Here's how you check for it
try:
    y = x / 0
except:
    print("Error occurred")

# And here you catch *only* the division by zero
try:
    y = x / 0
except ZeroDivisionError:
    print("You can't divide by 0!")
    print("Really, you can't!")
except TypeError:
    print("You can't divide these types!")
Points to remember

- Python variables and expressions have **types**
  - We can ask, e.g., `type(x)` to learn about x’s type.
  - There are functions to convert between types

- Loops are used to execute the same steps over and over:
  - **for** loops:
    - do something a **number of times**, e.g., by stepping through a sequence values
  - **while** loops (later):
    - do something until a **condition** becomes true
Points to remember

- Sometimes Python “crashes” during program execution
- We get an error message which helps to fix the error (read the message carefully!)
- We can use exception handling to deal with situations that might raise a runtime warning or error.