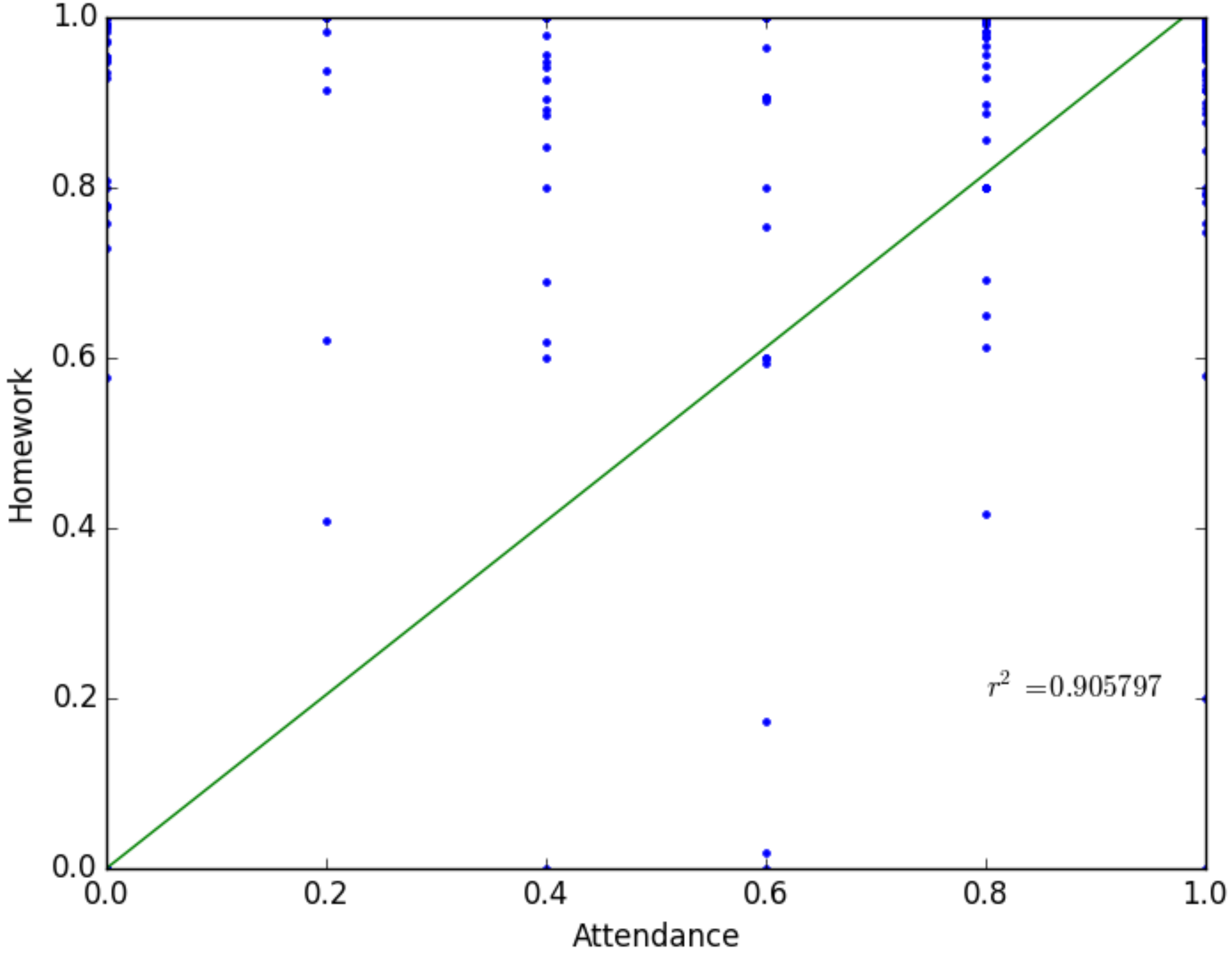
CS101: Intro to Computing Fall 2015

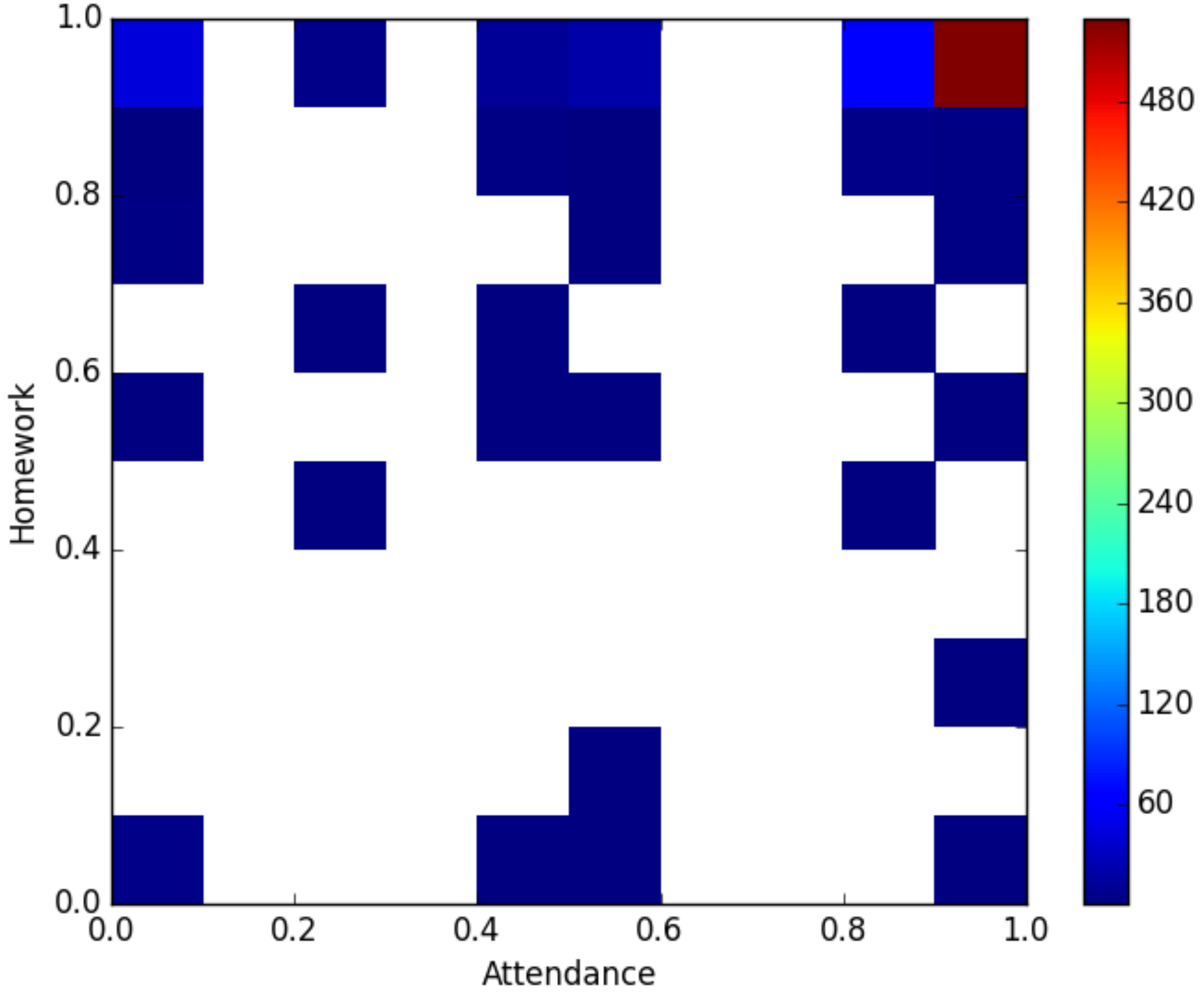
Lecture 9

Administrivia

- Homework 7 is due ***tonight***
- Homework 8 assigned (due on Mon)
- Midterm 1 is October 5th



Homework
Attendance
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
r^2 =0.905797



Homework
Attendance
0.0
0.2
0.4
0.6
0.8
1.0
60
120
180
240
300
360
420
480

REVIEW

```
s="ABCDEFGH"
t=""
i=0
while i<8:
    t=t+s[i+1]
    i=i+2
```

What is the value of t?

a)"ACEG"

b)"BDFH"

c)"ABCDEF"

d)"ABEF"

```
s="0123456789"
t=""
i=0
while i<5:
  if (i%2)==1:
    t=t+s[i-1]
  else:
    t=t+s[i+1]
  i=i+1
```

What is the value of t?

a) "92143"

b) "103254

c) "10325"

d) "921436"

LOOPING

While loop

- Allows for ***repeated execution*** of code
- Execute a block over and over as long as a Boolean condition is True
- ***Stop executing*** if Boolean condition is False

While loop

- We create an ***while loop*** by typing:

1. the keyword ***while***
2. a Boolean expression
3. a ***block*** of code

Accumulator pattern

- Common and useful pattern to design programs
- ***Accumulator*** variable keeps track of result
  - Updated in each loop iteration

Solution

```
def sum_digits(n):
  s=str(n)
  i=0
  result=0
  while i<len(s):
    result=result+int(s[i])
    i=i+1
  return result
```

FOR LOOPS

Example

```
i=0
while i<len(s)
  print s[i]
  i=i+1
```

For loop

- Loop construct to make our lives easier
- Used to iterate over *iterable* types
  - Example: strings (more to come)
- Step through a sequence “one at a time”

For loop

- We create an ***for loop*** by typing:

1. the keyword ***for***
2. a loop variable (just a variable)
3. they keyword ***in***
4. an iterable
5. a ***block*** of code

Example

```
my_string=“abcdefg”
for letter in my_string:
  print letter
```

Solution

```
def sum_digits(n):
  result=0
  for letter in str(n):
    result=result+int(letter)
  return result
```

```
s=“abcdefg”
t=“”
for c in s:
    t=c+t
```

What is the value of t?

a) “abcdefg”

b) “gfedcba”

c) “a”

d) “g”

```
s=“Run The Jewels”
t=“”
for c in s:
   if c.isupper():
      t=t+c.lower()
```

What is the value of t?

a) “RTJ”

b) “un he ewels”

c) “”

d) “rtj”

FILE INPUT

Files

- Iterable type
- Created with built in function `open()`
- 1 argument: file name as a string (for now)
- Each item in the iterable is a string representing one line in the file

```
for line in open(“words.txt”)
    print line
```

Example

```
total=0
for line in open(“numbers.txt”):
  n=int(line)
print total
```

Example

```
for w in open(“words.txt”):
  vowels=0
  for c in w.lower():
    if c in ‘aeiou’:
      vowels+=1
  print w.strip()+” %i” % vowels
```

LISTS

Lists

- Represents an ordered collection of *items* or *elements*
  - It’s iterable
- A ***container*** type
  - Contains other values of ***any type***
  - **NOTE***:* elements don’t have to be the same type

Lists

- We create an ***list*** by typing:

1. an open square bracket [
2. items of the list, separated by commas
3. a closing squre bracket]

Similarity to Strings

```
x=[10,3.14,"Ride"]
print x[1]
print x[1:3]
print len(x)
for i in in x:
  print i
```

Dissimilarity to Strings

- Strings are ***immutable*** (we can’t change the *contents* without ***creating a new string***)

```
s=“Puraty Ring”
s[3]=“i”  ← NOT ALLOWED
s=s[:3]+”i”+s[4:]
```

Dissimilarity to Strings

- Lists are ***mutable*** (we ***can*** change the contents of a list)

```
x=[4,1,2,3]
x[3]=-2 ← item assignment
x.append(5)
del x[1]
x.sort()
```

DANGER!! DANGER!!

- The *sort* and *append* methods modify the list ***itself***
- This means they **RETURN NULL**

```
x=[1,2,3,4]
x=x.append(5)
print len(x)
```

Range

- The ***range function*** returns a list of integers
- Two arguments:
  - the starting value our range
  - the ending (not included!) value in our range

```
x=range(2,5)
```

Example

```
total=0
for x in range(0,1000):
  total=total+x
print total
```

Example

```
total=0
for x in range(0,1000):
  prime=True
  for y in range(2,x):
    if (x%y)==0:
      prime=False
```

TUPLES

Tuple

- A tuple is an ***immutable*** sequence of any type
  - An immutable version of a list
- Literal: item in the tuple separated by commas (can add parentheses)

```
t=(1,3.14,”Hi”)
```

```
t=(1,3.14,”Hi”)
t[0:2]
t[-2]
len(t)
1 in t
t[2][1]
```

Tuple assignment

- A tuple can go on the ***left side*** of an assignment statement
- Allows us to make ***multiple assignment***s at once

```
one,pi,hello=(1,3.14,”Hi”)
```

- Convenient for swapping values:

```
x,y=y,x
```

Tuple return values

- A tuple can be used in a return statement
- Allows us to ***return multiple values*** at once

```
def fun():
   return (1,2,3)
```

- When calling, can use tuple assignment

```
a,b,c=fun()
```

String formatting with tuples

- We can use tuples on the ***right side*** of the string formatting operator
- Allows us to insert ***multiple values*** into the string

```
“%i %i %i” % (1,2,3)
```

```
s=???
x=10
y=“Hello”
z=3.14
print s % x,y,z
```

a)“%i %f %s”

b)“%f %s %i”

c)“%i %s %f”

d)None of the above.

Quadratic equation redux

MODULES

Modules

- A collection of Python specialized functions, variables, and even types
- We need to ***import*** the module

```
import math
```

- Can then access things within the module using ***attribute operator***

```
math.sqrt(math.pi)
```

From

- Can choose what to import with ***from***

```
from cmath import phase
phase(1+1j)
```

- Can then access things within the module using ***attribute operator***

```
math.sqrt(math.pi)
```

READABLE CODE

Our First Program

```
x = 10
y = x**2
y = y + y
print y
```

Writing readable code

- We should always strive to write code that is easy to read.
  - Our variables should have ***descriptive*** names.
  - We should also ***annotate*** our code.
- **REMEMBER**: A program is set of instructions a computer executes ***to achieve a goal***.

Commenting

- ***Comments*** are text that the interpreter ignores
- Comments are to help ***a person*** read our program
- The # symbol indicates a comment
  - Anything after that symbol is ignored

Abbreviated assignment

```
x=x+1
x+=1
```