

Lecture 18

Chapter 8 Input, Output, and Files

8.1 Standard Input and Output (review)

8.2 Formatted Strings

8.3 Working with Files

8.5 Case Studies

8.1 Standard Input and Output

The simplest form of receiving input from a user is through the use of the `raw_input` function.

Syntax: `raw_input(['prompt'])`

When called, the system waits while the user types in a string of characters. After the Enter key is pressed, the function returns the string of entered characters, up to but not including the final newline.

(The function reads a line from input, converts it to a string (stripping a trailing newline), and returns that.)

Another option is `input` function.

Syntax: `input(['prompt'])`

Input function uses `raw_input` to read a string of data, and then attempts to evaluate it as if it were a Python program, and then returns the value that results.

8.1 Standard Input and Output

The simplest form for generating output is `print` command.

This command accepts zero or more subsequent arguments separated by commas.

! Note: prior to Python 3.0 `print` was not formally a function, but instead a keyword of the language. Therefore its arguments were not specified within parentheses.

Several subtleties in the behavior of the command:

1. the arguments may be literals, identifiers, or compound expressions;
2. if an individual argument is not already an instance of the string class, it is automatically converted to a string;
3. when multiple arguments are given, an explicit space is automatically inserted between successive arguments.
4. by default, the `print` command generates one final newline character after printing all the arguments (which can be suppressed if a trailing comma is given following the last argument)

8.2 Formatted Strings

To output formatted strings we can use the `%` operator (modulo). `%` sign starts the conversion specifier.

Examples:

```
num,denom=3,4
print '%d / %d = %f'%(num,denom,float(num)/denom)
```

tuple

Produces: 3 / 4 = 0.750000

```
n=23
print '%4d'%n
```

type designator (conversion type)

Produces: 23

Underscores stand for two spaces that are added in front of 23.

```
print '%04d'%n
```

Produces: 0023

Number is padded with two leading zeros

8.2 Formatted Strings

In general: a conversion specifier contains two or more characters and has the following components, which must occur in this order:

1. The '%' character, which marks the start of the specifier.
2. **Mapping key** (optional), consisting of a parenthesised sequence of characters (for example, (somename)).

Example:

```
print '%(language)s' % {'language': "Python"}
```

produces: Python

8.2 Formatted Strings

3. **Conversion flags** (optional), which affect the result of some conversion types.

Flag	Meaning
'#'	The value conversion will use the “alternate form” (where defined below).
'0'	The conversion will be zero padded for numeric values.
'-'	The converted value is left adjusted (overrides the '0' conversion if both are given).
' '	(a space) A blank should be left before a positive number (or empty string) produced by a signed conversion.
'+'	A sign character ('+' or '-') will precede the conversion (overrides a “space” flag).

Example: `print '%04d'%n` produces: `0023`
↑
conversion flag

8.2 Formatted Strings

4. **Minimum field width** (optional). If specified as an '*' (asterisk), the actual width is read from the next element of the tuple in values, and the object to convert comes after the minimum field width and optional precision.

Example: `print '%04d'%n` produces: `0023`


5. **Precision** (optional), given as a '.' (dot) followed by the precision.

Example: `print '%7.3f'%1.2345`
produces: `__1.234`

8.2 Formatted Strings

6. **Conversion type.**

Conversion	Meaning
'd'	Signed integer decimal.
'i'	Signed integer decimal.
'o'	Signed octal value.
'x'	Signed hexadecimal (lowercase).
'X'	Signed hexadecimal (uppercase).
'e'	Floating point exponential format (lowercase).
'E'	Floating point exponential format (uppercase).
'f' or 'F'	Floating point decimal format.
'c'	Single character (accepts integer or single character string).
's'	String (converts any Python object using str()).

8.2 Formatted Strings

Example:

```
data = [1024, 4, 16, 32]
print '%4d ' * len(data) % tuple(data)
print '---- ' * len(data)
```

```
produces: 1024    4    16    32
          ----  ----  ----  ----
```

8.3 Working with Files

Programs must be able to read data from file and to write data to files. It is especially needed when we have a large volume of data.

Python supports a built-in class `file` to manipulate files on the computer.

Constructor of Python's file class accepts two parameters:

- *file name* (as string), and
- *access mode* (as string, optional)
 - r – for reading (default mode)
 - w – for (over)writing
 - a – for appending to the end of the file

Example:

```
file1 = file('inputData.txt')
```

-file inputData will be open in read-only mode

```
file2 = file('outputData.txt', 'w')
```

- file outputData will be open for writing (re-writing)

8.3 Working with Files

The syntax

```
file('sample.txt')
```

is equivalent in effect to the use of the built-in function

```
open('sample.txt')
```

that returns as associated file instance

Selected behaviors of Python's `file` class:

Syntax	Semantics
<code>open()</code>	Returns a file object (used with two arguments)
<code>close()</code>	Disconnects the file object from the associated file (saving it, if necessary)
<code>flush()</code>	Flushes the buffer of written characters, saving the underlying file
<code>read()</code>	Returns a string representing the (remaining) contents of the file
<code>read(size)</code>	Returns a string representing the specified number of bytes next in the file
<code>readline()</code>	Returns a string representing the next line of the file
<code>readlines()</code>	Returns a list of strings representing the remaining lines of the file
<code>write(s)</code>	Writes the given string to the file. No newlines are added.
<code>writelines(seq)</code>	Writes each of the strings to the file. No newlines are added.
<code>for line in f</code>	Iterates through the file <code>f</code> , one line at a time

Statistics Project

Example: Let's do the project **8.18-8.19**, p. 295-296

Write a program that reads a data set of two-dimensional points and calculates the “**line of best fit**” (or **regression line**) for that point set (namely, the line that minimizes the sum of the squares of the vertical distances between the points and the line)

You may assume that the file is formatted so that each line describes a single point, denoted by x and y coordinate values separated by a space.

The equation of the line is in the form $y=mx+b$, where m is the **slope of the line**, and b is the **y-intercept**. m and b can be computed using the formulae (given in the next slide)

Your program should plot the data, report the equation for the line, and draw the line.

Statistics Project

Example: Let's do the project **8.18-8.19**, p. 295-296

$y=mx+b$, where m is the slope of the line, and b is the y-intercept.

$$m = \frac{(n * (\sum x_i * y_i)) - (\sum x_i) * (\sum y_i)}{(n * (\sum x_i^2)) - (\sum x_i)^2}$$

$$b = \frac{(\sum y_i - m * \sum x_i)}{n}$$

Statistics Project

Example of input:

Content of the file data.txt:

```
12 23
14 45
-4 -20
-15 -60
15 40
17 60
-10 -48
```

Statistics Project

That's what we will do in statistics class:

	x	y		
	12	23		
	14	45		
	-4	-20		
	-15	-60		
	15	40		
	17	60		
	-10	-48		
sum	29	40	3986	1195

Statistics Project

That's what we will do in CSI32 class:

Assume that we were able to compile the following list:

```
data=[(12,23),(14,45),(-4,-20),(-15,-60),(15,40),(17,60),(-10,-48)]
```

Then we will do the following:

```
n=len(data)
```

```
sumxy, sumx, sumy, sumx2 = 0,0,0,0
```

```
for i in range(n):  
    sumxy += data[i][0]*data[i][1]  
    sumx += data[i][0]  
    sumx2 += data[i][0]**2  
    sumy += data[i][1]
```

At the end of this for loop, $sumx = 29$, $sumy = 40$, $sumxy=3986$,
and $sumx2=1195$

Statistics Project

Summary:

$$\text{sum}x = \sum x_i = 29, \quad \text{sum}y = \sum y_i = 40,$$

$$\text{sum}xy = \sum x_i * y_i = 3986, \text{ and } \text{sum}x^2 = \sum x_i^2 = 1195$$

And using, the given above formulas:

$$m = \frac{(n * (\sum x_i * y_i) - (\sum x_i) * (\sum y_i))}{(n * (\sum x_i^2) - (\sum x_i)^2)} = \frac{(7 * 3986 - 29 * 40)}{(7 * 1195 - 29^2)} = \frac{26742}{7524} \approx 3.55$$

$$b = \frac{(\sum y_i - m * \sum x_i)}{n} = \frac{(40 - \frac{26742}{7524} * 29)}{7} \approx -9.01$$

Statistics Project

Therefore the equation of the regression line ('best fit line') is

$$y = 3.55x - 9.01$$

Now, let's make a sketch of our program:

```
def main():
```

```
    # read data from the input file
```

```
    data=readData()
```

```
    # display data on the screen
```

```
    display(data)
```

```
    # find regression
```

```
    m,b=findRegression(data)
```

```
    # draw regression line
```

```
    draw_reg(m,b)
```

```
    # display the equation of the regression line
```

```
    dispEq(m,b)
```

```
main()
```

Statistics Project

`readData()` function

We need to read data from a given file and store it somewhere. For example, we can store the tuples (x,y) in a list.

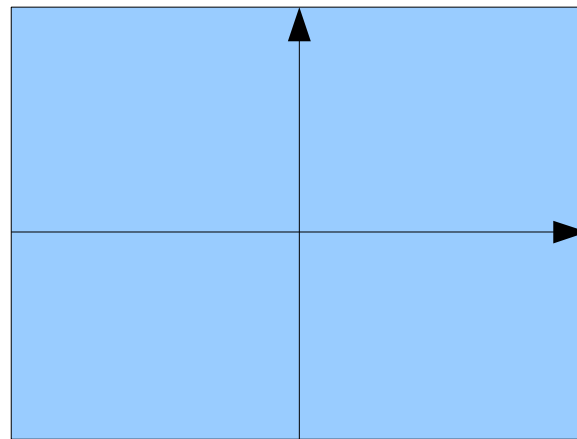
$[(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)]$

See the program [project18.py](#) for definition of this function.

Statistics Project

`display(data)` function

Then, we need to display the point in a graphical window. It means that we will need to draw x-axis and y-axis and plot the points in the corresponding places. A possible solution:



graphical
window

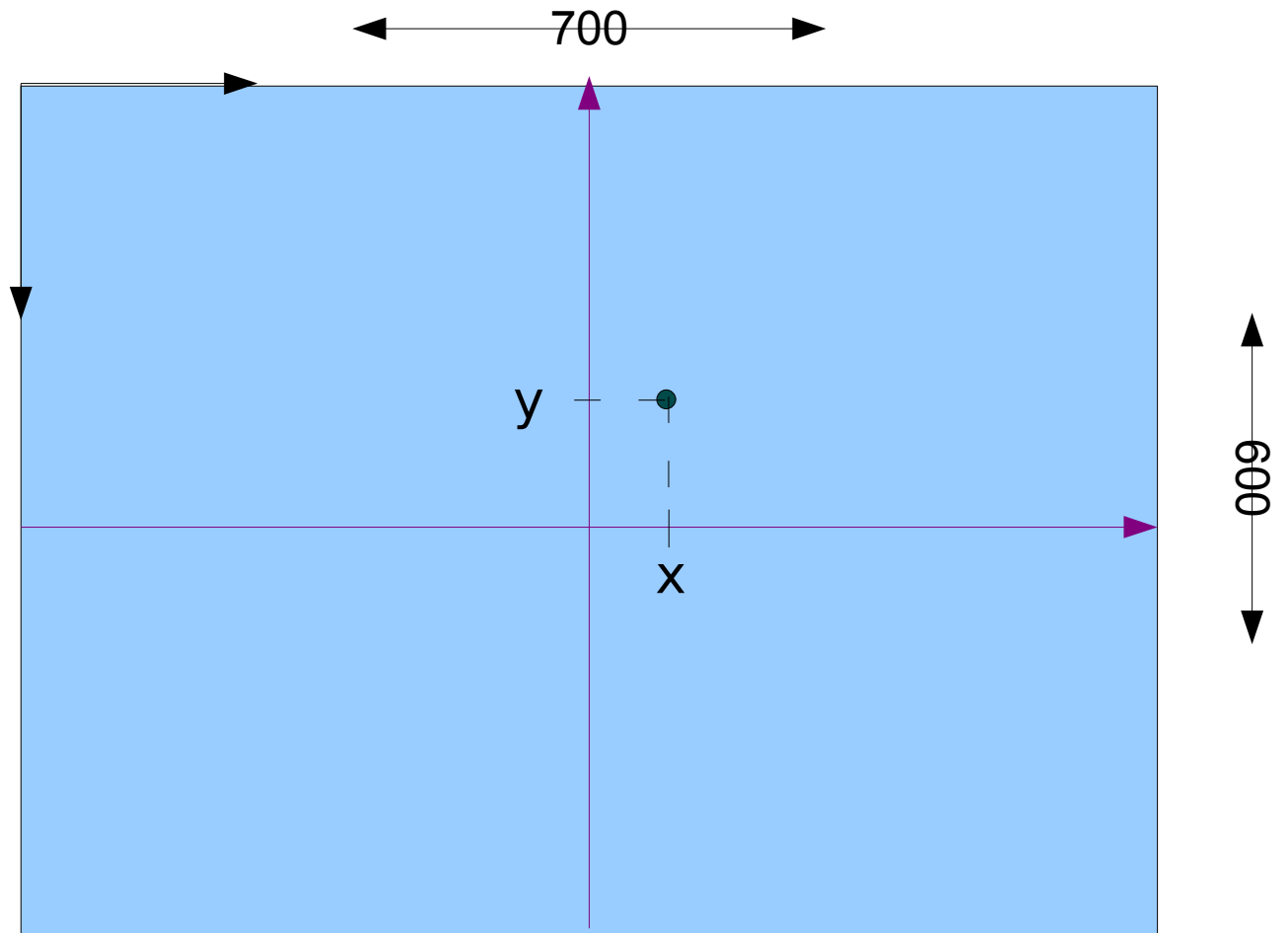
Recall, that the graphics window has different system of coordinates, therefore x- and y-coordinates of points in a regular rectangular coordinate system will have to be converted to points in the graphics window.

Statistics Project

`display(data)` function

Let's see how this conversion can be done. Assume that we have a fixed size of graphics window (700x600).

$$x' = x + 350$$
$$y' = -y + 300$$



Statistics Project

`findRegression(data)` function

This function will find the slope of the regression line (m) and the y-intercept of the line.

We will use the formulas and instructions that we had on slides 17-18.

See the definition of this function in [project18.py](#)

Statistics Project

`draw_reg(m, b)` function

This function will draw the regression line, given slope m and y-intercept b . We will use a path with two points.

Since the equation of the regression line is $y=mx+b$, all we need to do is use two values for x , find the corresponding values for y .

x	y
-100	$m*(-100)+b$
100	$m*100+b$

Let's use -100 and 100 for x -values. Don't forget to convert x - and y -values to the graphics window coordinate system (using the same formulas)

See the definition of the function in [project18.py](#)

Statistics Project

`dispEq(m,b)` function

This function will display the equation of the regression line in the graphics window. For that we will need to create an instance of class Text and move it to the left bottom part of the graphics window.

See the definition of this function in [project18.py](#)