Simple Input/Output

Data In/Data Out

- Programs are not very useful if they cannot communicate their results
- Programs also are not terribly useful if they cannot change their controlling parameters to compute different results
  - Note: always assume that there is no such thing as a "one off" program. Odds are very high that somebody will reuse it even if it seems to have a very limited purpose. Make the program read its input values, don't hard-code them

Files

- We read from or write to files.
- In Unix nearly everything, including the console, is a file. (Stdin, stderr, stdout are all files.)
- For now we will only deal with the standard streams

Reading Input

- We can ask the user to enter data.
- In some cases we can read data in the form of a command-line option

```
<interpreter> myprog --option <somevalue>
```

- The syntax is highly language specific
- We will learn to read from named files later

List-Directed IO

- List-directed IO allows the compiler or interpreter to format the data.
- Input
  - Python read from standard input (Python < 3.0)

    ```
    string=raw_input("Please enter the value:")
    ```

    Note that this writes to stdout and reads from stdin. Also note that it reads only strings regardless of what type you think the value is or should be. You will need to convert the type if the variable must be a number.

Output

- Every language has a way to print its results. Often the word is something like “print” or “write” with various combinations of other syntax.
- For now we will be printing to the console (standard output)
- Neat output is important for legibility, so in many cases we must *format* the output. The syntax here is also highly specific to languages.

List-Directed IO

- Output
  - Python

```
print var
print (var1,var2,var3)
```

The second form will be required for Python 3.0 and up.

Formatted Input/Output

With list-directed I/O, the compiler or interpreter decides how to input the data or how to arrange and organize the output.

Often we need or want to control the layout.

For this we need formatted I/O.

Edit Descriptors

- The edit descriptor modifies how to output the variables. They are combined into forms like
- RaF.w
- Where R is a repeat count, a is the descriptor, F is the total field width *including* space for +-, and if requested +-e and exponent, and w is the number of digits to the right of the decimal point.
- This is the basic pattern for many languages (some can't do the repeat, many use F.wa)
- Strings and integers take only F (RaF)

Python Formatted Output

- Python has decided to make things harder.
- Old way:

```
print “The value is %5.3f” % x
```

- New way (3.0 and up)

```
print 'The value of PI is approximately {}.'.format(math.pi)
```

You may use the old way for this class

General pattern is

```
print “format string” % variable
```

Must use a *tuple* (and the %) if more than one variable to be printed

```
print “format string” % (var1,var2,var3)
```

Python Format Strings

- These are formed similarly to other scripting languages and C
- The format string embeds the edit descriptor for the variables
- Common descriptors:
- i or d : integer (d for “decimal” not “double”)
- f : floating point
- eE, gG: e means scientific notation, g means “general” (interpreter chooses) withE/G printed lower/ upper case
- s string
- %F.w*a* or %*a* is the placeholder/formatter for a variable (*a* would be i/d, f, eE/ gG, s, whatever)

Python Example

- print “The answer is %12.8f” , my_result
- Also
- print “This answer is %.8f” , my_result