How to Write a Program

Writing a program is hard. For simple programs like the temperature converter we can just sit down at the computer, open a text editor, create a new document, and start typing. Most programs, however, require a little planning.

The process we recommend for Bi 410/510 involves three steps:

- write a **specification** of what the program should do
- write a **design** document that outlines how the program will work
- write the program, stopping often to test each new addition, using IPython to **experiment** as necessary

This document illustrates the process with a program that computes the GC content of a DNA sequence.

Project Spec

The first step is to figure out **what** the program will do

(later the design document will describe **how** the problem will be solved)

Specify what sorts of inputs the program expects and what it will produce as its outputs.

If possible write a set of examples that show how the program will be run and what the expected output will be.

Example: the spec for the “plural” program is to read one word from the command line and to print the plural form. The spec should show examples of the kinds of words the program should handle.

```
$ python plural.py duck
ducks

$ python plural.py fish
fishes
```

Specification for the GC content program

The program will be named gc_content.py

The input sequence will come from the command line, the output should be a percentage (a number between 0 and 100)

Examples of what we expect when the program is working:

```
$ python gc_content.py GATTACA
28.57 %

$ python gc_content.py AAAACCCCTTTTGGGG
50.0 %

$ python gc_content.py CCCCCCCCCC
100.0 %

$ python gc_content.py AAAAAAAAAA
0.0 %
```

Expert Advice: *it's a good idea to plan in advance for extreme cases — these are a good source of bugs*

Design Document

The document does not have to be long or formal

Often it's just a simple "to do list" or outline of the major parts of the program

Design of the GC content program

- Read the string from `argv[1]`, save it in a variable
- Use the `count` method to count the number of G's and C's
- Assume the rest of the letters are A's and T's.
- Compute P = (#G's + #C's) / length of `dna`
- P is the relative frequency of G or C — we need to multiply it by 100 to turn it into a percentage

Expert Advice: *Avoid the "blank canvas" syndrome.*

If you open a text editor to start coding you'll be faced with too many choices. A design document helps you get started.

As you gain more experience you might just open up a text editor and start typing a program as simple as this one.

Until then, write a short "design sketch", be as thorough as possible, anticipate any problems you might encounter

Sandbox

For this project the key is making sure we can count letters using the `count` method

It would be a good idea to experiment with the count method, try it out on some test data, and debug the expression that computes GC content

```
In [1]: s = 'GATTACA'

In [2]: len(s)
Out[2]: 7

In [3]: s.count('G')
Out[3]: 1

In [4]: s.count('C')
Out[4]: 1

In [7]: 2/7
Out[7]: 0.2857142857142857

In [8]: (s.count('G') + s.count('C')) / len(s)
Out[8]: 0.2857142857142857
```

Code

Given that design document it’s very easy to write the final program:

```
# Compute the GC content of a strand of DNA

# John Conery
# Jan 22 2018
# Usage:  python gc_content.py S

from sys import argv

dna = argv[1]

ng = dna.count('G')
nc = dna.count('C')

pct = (ng + nc) / len(dna)

print(round(pct * 100, 2), ‘%’)
```

Note the comments at the top of the program, including author name(s) and “usage string”

Expert Advice: *Make a program template — a file that has the header comments and import statement that are likely going to be in all your programs.*

The first step in the coding process is to make a copy of the template, revise it to fit the new project.

Iteration

You don’t have to complete all three steps (spec, design, code) in that order

In fact we recommend an “iterative” design process

- write the spec, maybe for a preliminary version that leaves out things you know will be in the final version
- write a design / to-do list for the preliminary version
- implement and test parts of the preliminary version — you might uncover problems or cases you didn’t anticipate
- update the spec and design as you go

Examples: adding more cases to the plural program, or modifying the GC content program to handle inputs that contain N or other letters

The ultimate goal: keep a record of what you did so that you (or someone else) can come back to the project in the future.

Template

Most programs follow a familiar pattern: import `argv`, get values from `argv`, print results.

We've put together a template that you can download and use if you want

Type this command to see the contents of the template:

```
$ curl pages.uoregon.edu/conery/Bi410/template.py
```

Use "redirection" to save the template:

```
$ curl pages.uoregon.edu/conery/Bi410/template.py > prog.py
```