Homework 2 - Variable and Functions

September 5, 2018

1 Chapter 2 Homework - Variables, expressions and statements

This homework corresponds to the material covered in Chapter 2 of *Think Python*, and tries to reinforce the material contained therein, as well as its assocated lecture(s). You should have read and understood Chapter 2 in the textbook before embarking upon this assignment.

1.1 Addendum to Section 2.2 -- Variable names

For the most part, you should be sticking with variable names that are composed of letters (and mostly lower case, at that), non-leading digits and non-leading underscores. This isn't to say that the above are hard-and-fast rules, but exceptions to this guidance tends to attribute special meaning to variables which you may not intend. Some of these "exceptions" will be addressed below.

- Python purists follow the *convention* that variable names that "want" to be multiple words use underscores to separate the words. For example, `acceleration_due_to_gravity`. The other convention is what's called *camel case*, named because it lower at the ends with capitalization humps in the middle. which is used in other languages such as Java. This variable would be `accelerationDueToGravity` Either is acceptable, but whichever style you ultimately choose should be the same as that used in code around it. For example, if the original author of the code that you've written went with the underscore style, your modifications should follow that style.
- Meaningful variable names are **massively** important. If someone with whom your code is intended to be shared cannot readily understand the intent of your program, **you're not doing it right**. This is arguably one of the most important parts of programming. It only takes a tiny bit more effort to come up with names that make sense. Clear names are self-explanatory and help to minimize the need to document the explictly document the code with extra clarifying comments.

There are three "exceptions" to this rule that aren't *really* exceptions, but are also conventions.

You may come across three that have decades of use in computer science history and folklore -- `foo`, `bar` and `baz` (there are a few more less-frequently used ones, but these are the biggies). They might as well have been called `whatever` or `dude`. They are only used in contexts in which their meaning is irrelevant, like very narrow examples, for example `foo = foo + 1` to demonstrate how to increment the value of a variable. Recognizing (and using) these varaibles in the right context earn you some street cred among more-experienced programming nerds, but are never

intended to be used in anything resembling a "real" program. Whenever you see them, consider yourself in on the (very old) joke.

The second class of seemingly anemic variable names are ones like `i, j, k, x, y, f, s, n and c` or `ch`. At first these look like flagrant violations of the rules, but they're also decades-old conventions, but they actually do convey a bit of meaning. However, all tend to get used in short pieces of code with very narrow visual scope. `i, j` and `k` tend to be used as inconsequential integer variable names for walking through a list or sequence of items (and you may not even care what their values are). You may/will encounter them later on in the course when lists and arrays are discussed.

Similarly, `x`, `y` and `f` tend to stand for throw-away real (floating-point) values. `s` (or sometimes `str`) is usually used to briefly hold a string, `n` to hold some bounding number (similar to the common use of `N` in algebra or statistics).`c` or `ch`tend to get used to denote a single character value.

One might ask, why not use `index` for the name instead of `i`? Well, for the same reason that statistics people use *i* as the subscript in sumnations, or *x* as iterators in integrals. Longer names just aren't important, because it's implicitly understood that they're only used *right here*, so fancy naming is overkill.

Lastly, if your code is being written for an audience that has their own vocabulary of common acronyms and abbreviations, then these less elaborate names are actually the clearer ones. For example, if physicists are overwhelmingly your target audience, then `g, m, a` and `v`, in context, is highly likely to mean "acceleration due to gravity", mass, acceleration and velocity. Other longer names actually go *against* convention.

At the other extreme, punishingly long names can result in unattractively dense-looking code that can be even less welcome than single-character variable names, because it requires more effort to read. You can also create program errors simply by misspelling long variable names when spelled correctly elsewhere.

Other conventions that you should be aware of and should follow are that *constant* variables (seemingly an oxymoron) are often spelled in all uppercase. This convention is used in many languages, and to a fair extent, is used in Python. If you see a variable named MAX_VELOCITY, or MAX_STRING_LEN, you'll probably find it set in one place and will never change.

Knowing and following conventions free your brain up for thinking about important things rather than having to reason about notions that you can make assumptions about.

In this course you should strive for clarity because your audience is not just yourself but also your fellow students, project partners and (especially!) your instructors and graders. Conservatively named variables are always going to be OK, but **smarter**, shorter variable names are something you should always shoot for.

As stated in the textbook, *reserved* words used by the Python language are invalid variable names. By the end of the course you should know what most of them are and will learn to naturally avoid them.

1.1.1 Exercise #1

Which are valid, invalid, or inadvisable (and if so, why)? (Ex: my_dogs_name - valid)

a
acc
_acc_earth
accelleration
pos_x
position_x

xpos
x
xPosition
i
I
index
indexes
is
iterator
g
earth_gravity
gravity_earth
out_of_range
4th_of_July_is_on_Sunday
password
s
string
N
None
RUNNING_TOTAL
numberofsecondsuntilthenextpresidentialelection
number_of_seconds_until_the_next_presidential_election
secs_to_election

1.2 Addendum to Section 2.1 -- Assignment statements

A variable can be used to store a value of any type, even if it was formerly used to hold a value of a different type. In some sense, it is as if a variable stores both its own type and value simultaneously (because it does!).

In general, using the same variable to store values of different types in different places in a program is a bad practice, because it complicates knowing which operations may be performed on a given variable at all points in time. For your purposes, just use another variable, because that's usually what makes sense.

1.2.1 Exercise #2

For the assignment expressions below, which are invalid? List them in the same order they are presented below, and explain why they are not valid.

```
n = 1
n=2
n = -3
n=-4
n = +5
n = --6
n = n
0 = n
x = 3.4
```

```
x = 0.5
x = 00.5
x = .6
x = 7.
x = .0
x = -1.
x = --2.2
s = "Kermit"
s = 'Kermit'
s = `Kermit`
s == "Kermit"
s = "Kermit'
s = ""Kermit" the Frog"
s = "'Kermit' the Frog"
s = 'It's been a long, long time.'
s = "Ke
rmit"
s = "Ke\
rmit"
s = "4th_of_July"
4th_of_July = s
"s" = "ess"
b = 3
a = b = 3
b = a
b = c
a = b = 42
a = 14 = 42
```

1.3 Addendum to Section 2.3 -- Expressions and statements

Note that *expressions* result in values being printed out in interactive mode. This is not the case in script mode, as should become clearer below.

1.4 Addendum to Section 2.4 -- Script mode

Script mode is merely the mode where all of your code resides in a file -- a plain 'ol text file. You've already figured-out what all the steps are that your program needs, some of which you played with in interactive mode, and you just want to execute the entire program and get out the results. If the program requires no interaction, you can even directly launch it by attaching it to an icon on your desktop, or Start Menu entry, if you (or your mom) feel the need to set it up this way.

However, the classic way to run a Python program is by using your operating system's Command Prompt (or Terminal window, or Console, or whatever your OS calls it). If your program text was in a text file named myprog.py, you can execute it with "python3 myprog.py". If the program is interactive and reads from the keyboard or prints out messages, they'll be dealt with through that window.

Unlike expressions entered in interactive mode, expressions' results are *not* printed out when executed in script mode. Why this is should be clear in the next few weeks. If you need to print

out the value of anything, you use the `print()` function, as was briefly mentioned in this chapter (don't worry, you'll learn much more about functions in a week or two).

You will learn more about script mode in this week's lab.

1.5 Addendum to Section 2.5 -- Order of operations

I fully agree with the author about not spending too much time learning the order of operations for all of Python's operators. When writing code, it's almost always best to use parentiesis to show your intentions. The problem is when you're reading someone else's code and they're not as considerate. When necessary I google for "python operator precedence" and annotate the expression in question with parenthesis of my own.

Even an expression as simple as

```
x = a + b * c
```

should still generally be written as

```
x = a + (b * c)
```

1.5.1 Exercise #3

In preparation for not passing this class, your class project partner has written some lame code that contains the following statement(s). Re-write it so that it contains parenthesis corresponding to the order of operations specified in the language.

```
y = a * x ** 2 + b * x ** 1 + c - d / k * x ** -1
```

1.5.2 Exercise #4

Do the end-of-chapter Exercise 3.1 in Section 3.14 from *Think Python*.

1.5.3 Exercise #5

Write a function called `my_sin(degrees)` (not radians!) that uses the first three terms of a Taylor series expansion to approximate the value of *sine(x)*. It should return a floating-point number.

For your convenience, the formula for this function is *sin(x) = x - (x^3)/3! + (x^5)/5! - (x^7)/7! + (x^9)/9! - ...*