**syntax** The form in which a language is written. Different programming languages and different human languages have different syntaxes. A statement that is syntactically invalid can't be interpreted. Human languages tend to have relatively flexible syntax, so I could write "Very poorly written this sentence is." Even though this is invalid syntax in English, you can understand it. Human languages trend also to have an ambiguous syntax. So if I write "I was driving down the road in my car." It is ambiguous whether the road is in my car, or I'm in my car. You can guess what i mean because you know roads don't belong in cars. Programming languages are designed to not have this kind of ambiguity, and as a result are unforgiving of syntax errors.

**semantics** The semantics is what something means. In human languages (see the car example advice) we often use semantics to resolve an ambiguity in syntax. Semantics in programming languages is important, because it reflects what you want the program to do. Until you know what you want the program to do, you aren't going to be able to write the program, but new programmers can get caught up in the syntax and lose docks on what the semantics of their goal are.

**pseudocode** A description of what a program should go in full detail, but without necessarily the syntax of a programming language. Pseudocode is to be yeah by humans, not computers, and is designed to capture the desired semantics, without the bother of getting the syntax right. There do exist formal syntaxes for pseudocode (for example for writing a computer science paper) but we aren't going to bother with anything like that, and will default to writing pseudocode that looks vaguely like Python.

I'll now introduce a few of the *kinds* of semantics you'll be using in python. I'll try to use pseudocode that pretty well matches with Python itself.

**variable assignment** You encountered this in `https://paradigms.oregonstate.edu/activity/834`. It means to compute the thing on the right and then give it the name on the left.

```
volume = area*height
```

Remember that in this statement we pronounce `=` as "gets". Sometimes it is written in pseudocode like

`volume` ← `area*height`

to reflect that you are computing the right hand side, and then storing the result in the variable named on the left.

**expressions** An expression is something that you can compute that has a value. So the thing `area*height` is an expression.

**comparison** This is a kind of expression which has a true/false (or *boolean*) value. In Python we write an equality comparison as `==`, so a comparison would be something like `volume == 10`.[1] Comparisons can also be $\leq$, $<$, etc.

**if/else statement** This is a way to make what your code does depend on a comparison. It reads much like an English sentance:

[1] Yes, I'm getting into syntax here, but you also can't write clear pseudocode without understanding the semantic distinction between assignment and comparison.

```
if volume > 10:
    print('your house is big')
else:
    print('your house is cozy')
```

This would print a comment on the size of your house based on the value of the variable `volume`.

**block** A sequence of statements (or things to do). We saw blocks above with the `if` statement, but only did one thing. We could do more.

```
if volume > 10:
    print('your house is big')
    cost = volume*100 # big houses have economy of scale
else:
    print('your house is cozy')
    cost = volume*200
print('the cost is', cost)
```

In many other programming languages blocks are indicated with curly braces

```
if volume > 10 {
    print('your house is big')
    cost = volume*100 # big houses have economy of scale
} else {
    print('your house is cozy')
    cost = volume*200
}
```

but python uses indentation and a preceding colon to delimit a block.

**while** A "looping" construct that causes a block to be executed as long as a comparison is true.

```
while volume > 10:
    print('volume is still too big')
    volume = volume - 1
```

With practice you can reason your way through while loops, but that often takes some time, which often makes your code less legible.

**for** Another looping construct which causes a variable to successively take a number of values. This is the place where for now we will let our pseudocode differ from python, since it can be a little hard to guess what values python will use, and there are many ways to specifiy those values

```
for length in integer values starting with 1 and ending with 10:
    if length*(length-1) == 56:
        print('The answer is', length)
```