Acknowledgement: This tutorial is a modified version of Melanie Walsh's tutorial. 🙏

⚠️ Before you start ⚠️

Duplicate this Jupyter Notebook in your `week-02` folder (right-click -> Duplicate) and then your last name to the beginning of it (ie. `blevins-python-variables.ipynb` - otherwise you risk having all your work overwritten when you try to sync your GitHub repository with your instructor's repository.

⚠️ *No, seriously: check the name of this file. Is it the copy you made? (ie. `blevins-python-variables.ipynb`). If so, you can proceed* ⚠️

Variables

Variables are one of the fundamental building blocks of Python (or any coding language). A variable is like a tiny container or box 🎁 where you store things, such as filenames, words, numbers, collections of words and numbers, and more. You can then take these boxes and move them around, send them places, perform actions on whatever is inside it, etc.

Assigning Variables

Each variable needs to be "assigned" - ie. given a name. This is the equivalent of adding a label to the outside of the box. The label itself doesn't change what is inside the box, but generally speaking you want it to be descriptive of what *type* of stuff is inside it. So if I added a cake to the box, the label might be `food` or `dessert`.

You assign variables with an equals `=` sign. In Python, a single equals sign `=` is the "assignment operator," whereas a double equals sign `==` is the "real" equals sign for doing addition, subtraction, etc.

In [8]:
```
some_variable = 100
print(some_variable)
```

```
100
```

In [9]:
```
2 * 2 == 4
```

Out[9]:
```
True
```

In [10]:
```
different_variable = "I'm the contents of another variable!"
print(different_variable)
```

```
I'm the contents of another variable!
```

Working With Variables

Variables are like wrapped boxes 🎁 with labels 🏷️ on them. When you create a variable named "author", imagine putting a label that says "author" on a box - you can put any value inside that box

(like "Shakespeare" or "Austen"), and whenever your code needs to know what's in that box, it just looks at the label and opens it up. The cool part is you can change what's inside the box at any time - if you put a different author's name in there, any part of your code that looks for the "author" box will automatically get the new value instead. You don't have to go around updating every place that uses that author's name; they all just look inside the same box.

Take the following example: let's say I wanted to have Python print out a sentence "I saw 5 blue cats dancing on the roof!" One way would be to write a print statement like this:

In [14]:
```
print("I saw 5 blue cats dancing on the roof!")
```

```
I saw 5 blue cats dancing on the roof!
```

I could also assign some variables to print the same sentence. Note: the `f` and `{}` code is just a way to print the contents of variables all in one line. Don't worry too much if you don't understand that.

In [16]:
```
animal = "cat"
color = "blue"
number = 5

print(f"I saw {number} {color} {animal}s dancing on the roof!")
```

```
I saw 5 blue cats dancing on the roof!
```

The benefit of this approach is that it's easy to update those variables if what I'm seeing on the roof keeps changing. (In this scenario I'm obviously taking advantage of Denver decriminalizing psilocybin). The type animal, color, or number might change, but I don't have to rewrite the sentence every time.

Try changing each of the variable `values` (ie. the contents their box - what is immediately after the `=`) below and run the cell to see how it changes the output).

In [19]:
```
animal = "cat"
color = "blue"
number = 5

print(f"I saw {number} {color} {animal}s dancing on the roof!")
```

```
I saw 5 blue cats dancing on the roof!
```

Now, this example might not save us much time. But what's nice about variables is that you can change their contents without having to rewrite all your code, and you can also apply actions to those variables. Let's add a couple lines of code:

In [21]:
```
animal = "cat"
color = "blue"
number = 5

print(f"I saw {number} {color} {animal}s dancing on the roof!")
print(f"Every time I see one {color} {animal}, I shout out (because I'm annoying)...")
for i in range(number):
    print(f"Wow, look!! It's a {color} {animal}! How many times have I said this? Oh rig
```

```
I saw 5 blue cats dancing on the roof!
Every time I see one blue cat, I shout out (because I'm annoying)...
Wow, look!! It's a blue cat! How many times have I said this? Oh right, 1x!
Wow, look!! It's a blue cat! How many times have I said this? Oh right, 2x!
Wow, look!! It's a blue cat! How many times have I said this? Oh right, 3x!
Wow, look!! It's a blue cat! How many times have I said this? Oh right, 4x!
Wow, look!! It's a blue cat! How many times have I said this? Oh right, 5x!
```

If I were **NOT using variables**, here is what the Python code would look like to get this output:

In [23]:
```
print("I saw 5 blue cats dancing on the roof!")
print("Every time I see one blue cat, I shout out (because I'm annoying)...")
print("Wow, look!! It's a blue cat! How many times have I said this? Oh right, 1x!")
print("Wow, look!! It's a blue cat! How many times have I said this? Oh right, 2x!")
print("Wow, look!! It's a blue cat! How many times have I said this? Oh right, 3x!")
print("Wow, look!! It's a blue cat! How many times have I said this? Oh right, 4x!")
print("Wow, look!! It's a blue cat! How many times have I said this? Oh right, 5x!")
```

```
I saw 5 blue cats dancing on the roof!
Every time I see one blue cat, I shout out (because I'm annoying)...
Wow, look!! It's a blue cat! How many times have I said this? Oh right, 1x!
Wow, look!! It's a blue cat! How many times have I said this? Oh right, 2x!
Wow, look!! It's a blue cat! How many times have I said this? Oh right, 3x!
Wow, look!! It's a blue cat! How many times have I said this? Oh right, 4x!
Wow, look!! It's a blue cat! How many times have I said this? Oh right, 5x!
```

Now, imagine I started seeing red dogs instead of blue cats. If I had NOT been using variables and typing out all of these print statements individually, I would have needed to make 14 different replacements!! (replacing `dog` for `cat` 7 times and replacing `red` for `blue` another 7 times).

With variables, all I have to do is update it once, when I'm assigning the value of the two variables:
`animal = "dog"`
`color = "red"`

Furthermore if I had seen 10 cats instead of 5 cats, I wouldn't need to add 5 more lines of print statements. Again, I can just change it once when I'm assigning variables:
`number = 10`

In [25]:
```
animal = "dog"
color = "red"
number = 10

print(f"I saw {number} {color} {animal}s dancing on the roof!")
print(f"Every time I see one {color} {animal}, I shout out (because I'm annoying)...")
for i in range(number):
    print(f"Wow, look!! It's a {color} {animal}! How many times have I said this? Oh rig
```

```
I saw 10 red dogs dancing on the roof!
Every time I see one red dog, I shout out (because I'm annoying)...
Wow, look!! It's a red dog! How many times have I said this? Oh right, 1x!
Wow, look!! It's a red dog! How many times have I said this? Oh right, 2x!
Wow, look!! It's a red dog! How many times have I said this? Oh right, 3x!
Wow, look!! It's a red dog! How many times have I said this? Oh right, 4x!
Wow, look!! It's a red dog! How many times have I said this? Oh right, 5x!
Wow, look!! It's a red dog! How many times have I said this? Oh right, 6x!
Wow, look!! It's a red dog! How many times have I said this? Oh right, 7x!
Wow, look!! It's a red dog! How many times have I said this? Oh right, 8x!
Wow, look!! It's a red dog! How many times have I said this? Oh right, 9x!
Wow, look!! It's a red dog! How many times have I said this? Oh right, 10x!
```

Variable Names

In the above example we named our variables `animal`, `color`, `number`.

Variable names can be as long or as short as you want, and they can include:

- ✅ upper and lower-case letters (A-Z)
- ✅ digits (0-9)
- ✅ underscores (_)

However, variable names *cannot* include:

- ❌ other punctuation (-.!?@)
- ❌ spaces ( )
- ❌ a reserved Python word

Changing the name of the variable does not change the content of the variable, it's just a label. So instead of `animal`, we could have named that variable `furry_friend`:

In [29]:
```
animal = "cat"
furry_friend = "cat"
print(furry_friend)
```

```
cat
```

In fact, we could literally call the variable just about anything we want (like `pancakes` or even a single letter like `f`). However, it's **best practice** to use a descriptive name for the variable that points towards what kind of content it has inside it.

In [31]:
```
pancakes = "cat"
f = "cat"
print(pancakes)
print(f)
```

```
cat
cat
```

Striving for Good Variable Names

As you start to code, you will almost certainly be tempted to use extremely short variables names like `f`. Your fingers will get tired. Your coffee will wear off. You will see other people using variables like

`f`. You'll promise yourself that you'll definitely remember what `f` means. But you probably won't.

So, resist the temptation of bad variable names! Clear and precisely-named variables will:

- make your code more readable (both to yourself and others)
- reinforce your understanding of Python and what's happening in the code
- clarify and strengthen your thinking

Example Python Code ❌ With Unclear Variable Names❌

For the sake of illustration, here's some of our same word count Python code with poorly named variables. The code works exactly the same as our original code, but it's a lot harder to read **as code**.

In [36]:
```
f = "cat"
x7gh = "blue"
banana_phone = 5

print(f"I saw {banana_phone} {x7gh} {f}s dancing on the roof!")
print(f"Every time I see one {x7gh} {f}, I shout out (because I'm annoying)...")
for i in range(banana_phone):
    print(f"Wow, look!! It's a {x7gh} {f}! How many times have I said this? Oh right, {i
```

```
I saw 5 blue cats dancing on the roof!
Every time I see one blue cat, I shout out (because I'm annoying)...
Wow, look!! It's a blue cat! How many times have I said this? Oh right, 1x!
Wow, look!! It's a blue cat! How many times have I said this? Oh right, 2x!
Wow, look!! It's a blue cat! How many times have I said this? Oh right, 3x!
Wow, look!! It's a blue cat! How many times have I said this? Oh right, 4x!
Wow, look!! It's a blue cat! How many times have I said this? Oh right, 5x!
```

Our original Python Code ✨ With Clearer Variable Names ✨

In [38]:
```
animal = "cat"
color = "blue"
number = 5

print(f"I saw {number} {color} {animal}s dancing on the roof!")
print(f"Every time I see one {color} {animal}, I shout out (because I'm annoying)...")
for i in range(number):
    print(f"Wow, look!! It's a {color} {animal}! How many times have I said this? Oh rig
```

```
I saw 5 blue cats dancing on the roof!
Every time I see one blue cat, I shout out (because I'm annoying)...
Wow, look!! It's a blue cat! How many times have I said this? Oh right, 1x!
Wow, look!! It's a blue cat! How many times have I said this? Oh right, 2x!
Wow, look!! It's a blue cat! How many times have I said this? Oh right, 3x!
Wow, look!! It's a blue cat! How many times have I said this? Oh right, 4x!
Wow, look!! It's a blue cat! How many times have I said this? Oh right, 5x!
```

Off-Limits Names

The only variable names that are off-limits are names that are reserved by, or built into, the Python programming language itself — such as `print`, `True`, and `list`.

This is not something to worry too much about. You'll know very quickly if a name is reserved by Python because it will show up in green and often give you an error message.

In [41]:
```
True = "cat"
```

```
  Cell In[41], line 1
    True = "cat"
    ^
SyntaxError: cannot assign to True
```

Your Turn

"My Favorite Things" Exercise:

- Insert a new code cell
  - Create three variables that hold information about your favorite book/movie:
    - One for the title
    - One for the year it was released
    - One for the main character's name
  - Then use these variables to print out two different sentences about your favorite book/movie
- Now create a **second code cell**. Use the same code as the first cell, but change the values of the variables to be about a different book/movie. Use the same print statements to see how your variables have changed between the two code cells.

Submit Your Jupyter Notebook

Once you've finished, please submit this tutorial to Canvas (note: this doesn't "count" as an official homework, it's just to practice submitting material on Canvas and to nudge you to complete the tutorial). Follow the instructions I've made for submitting homework and then submit your files to the corresponding assignment page on Canvas.

- Save your notebook
- `Kernel` -> `Restart Kernel and Run All Cells`
- `File` -> `Print` and try to find an option to Save/Print to PDF. Depending on your operating system and browser, this might be `Destination` -> `Save as PDF`, `Select Printer` -> `Microsoft Print to PDF` (instructions for different browsers). Name the file with the same naming convention as your .ipynb file (ie. `hw-01-yourlastname.pdf`) and save the resulting PDF file (ending in `.pdf`) into the same folder.

Note: use the naming convention `yourlastname-jupyter-notebooks-intro.ipynb` & `yourlastname-jupyter-notebooks-intro.pdf`. This will help us troubleshoot the pipeline for submitting to Canvas and make sure that this pipeline also works on my end.

In [ ]:
```
```