Variables

So what if you wanted Python to remember a certain value? `print` will output stuff onto the screen, but the stuff on the screen is not stored into memory: you can't reuse it. If you wanted to re-use a value or just in general, have Python remember something, we use something called variables. You can refer to a variable as a box of memory: it has a name so Python knows which variable is which, and it stores some value inside of it.

Variable Assignment

Now how do you make one? You name it, and you assign it something (usually a default value or an actual value). The usual format goes as follows: `name = value`. You can read this as the value stored inside the variable named "name" is equal to the value represented by "value". Here's a few examples of how this is done:

In [ ]:
```
# Example 2-1: Variable Assignment
name = "Anthony"
age = 4 * 5

print(name)
print(age)
```

```
Anthony
20
```

Note that you can also change the values of variables with this! If the variable already exists, then Python will simply update that variable with the new value you gave it:

In [ ]:
```
# Example 2-2: Reassigning Variables
name = "Anthony"
print(name)
name = "Tony"
print(name)
name = 1 * 2
print(name)
```

```
Anthony
Tony
2
```

Sequential Steps

Notice that the code above went in order: the first line executed, and then the second line executed, etc. Python executes code line-by-line starting from the top and going down. This means that the order in which you write your code matters! Keep this in mind when making your code (you may accidently `print` too early or too late. What if I moved all my `print` statements up by one line?)

In [ ]:
```
# Example 2-3: Disaster
print(hobby)
hobby = "Anthony"
print(hobby)
hobby = "Tony"
print(hobby)
hobby = 1 * 2
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[6], line 2
      1 # Example 2-3: Disaster
----> 2 print(hobby)
      3 hobby = "Anthony"
      4 print(hobby)

NameError: name 'hobby' is not defined
```

As seen with Example 2-3, you get an error because I tried printing out a variable before it was defined.

Naming Convention

You can output variables as well with `print`: the values inside the variable will be outputted. Now, here comes another problem: what can we name our variables? Python has some rules you need to follow in regards to naming convention:

- Variables can start only with a letter or an underscore
- Variables can containly only letters, numbers, and underscores
- Variables cannot be the same as a reserved keyword (these are words with special meanings built into Python)

Every seasoned programmer will also tell you to name your variables appropriately such that the name of the variable immediately tells

people (not just yourself) what the variable is used for. This prevents confusion if you ever look back at your code someday and saves others time trying to decipher your code. In other words, if someone random can't make a somewhat accurate guess of what value your variable should hold by just looking at the name, you need a better name. Obviously context does matter: if you're doing a math equation then `x` can be valid, but naming three numbers `a`, `aa`, and `aaa` generally is a very bad idea.

Math Assignment Operators

If you want to apply some basic math operation to a variable, you can directly do it with a math assignment operator. These operators are as follows:

- Addition (`+=`)
- Subtraction (`-=`)
- Multiplication (`*=`)
- Division (`/=`)
- Power (`**=`)
- Floor Division (`//=`)
- Modulus (`%=`)

All we did was add the operator to the front of the assignment operator! Now the variable will have the operation applied to it using the right-hand value to compute. For example, assuming we have `x = 2`, then `x += 3` is the same as `x = x + 3` or `x **= 3` is the same as `x **= 3`. You can visually see that the right hand value is used as the second number (or in other words, always on the right side of the operator) and the variable itself is always the left hand value (or on the left of the operator). This shorthand is very neat especially for longer calculations.

Examples of these operators can be seen in Example 2-4.

In [ ]:
```
# Example 2-4: Math Assignment Operators
number = 1

number += 2     # number is now 3
print(number)
number -= 1     # number is now 2
print(number)
number *= 3     # number is now 6
print(number)
number /= 2     # number is now 3.0
print(number)
number **= 2    # number is now 9.0
print(number)
number //= 2    # number is now 4.0
print(number)
number %= 3     # number is now 1.0
print(number)
```

```
3
2
6
3.0
9.0
4.0
1.0
```