Variable Assignment!

Some important rules to follow while naming variables:

1) Names cannot start with a number.

2) There can be no spaces in the name, use _ instead.

3) Can't use any of these symbols:'",<>/?|\()!@#$%^&*~
-+

4) Avoid using words that have special meaning in Pyth
on like "list" and "str".

Dynamic Typing

Python uses *dynamic typing*, meaning you can reassign variables to different data types. This makes Python very flexible in assigning data types; it differs from other languages that are *statically typed*.

```
In [1]: my_dogs = 2

In [2]: my_dogs
Out[2]: 2

In [3]: my_dogs = ['Sammy', 'Frankie']

In [4]: my_dogs
```

Pros and Cons of Dynamic Typing…

Pros of Dynamic Typing

- Very easy to work with
- Faster development time

Cons of Dynamic Typing

- May result in unexpected bugs!
- You need to be aware of `type()`

Assigning Variables

Variable assignment follows `name = object`, where a single equal's sign `=` is an **assignment operator.**

```
In [5]: a = 5

In [6]: a
Out[6]: 5
```

Here we assigned the integer object `5` to the variable name `a`.
Let's assign `a` to something else:

```
In [7]: a = 10

In [8]: a
Out[8]: 10
```

You can now use `a` in place of the number `10`:

```
In [9]: a + a
Out[9]: 20
```

Reassigning Variables

Python lets you reassign variables with a reference to the same object.

```
In [10]: a = a + 10

In [11]: a
Out[11]: 20
```

There's actually a shortcut for this. Python lets you add, subtract, multiply and divide numbers with reassignment using `+=`, `-=`, `*=`, and `/=`.

```
In [12]: a += 10

In [13]: a
Out[13]: 30
```

```
In [14]: a *= 2

In [15]: a
Out[15]: 60
```

Determining variable type with `type()`

You can check what type of object is assigned to a variable using Python's built-in `type()` function.

Common data types include:

1) **int** (for integer)
2) **float**

3) **str** (for string)
4) **list**
5) **tuple**
6) **dict** (for dictionary)
7) **set**
8) **bool** (for Boolean True/False)

```
In [16]: type(a)
Out[16]: int

In [17]: a = (1,2)

In [18]: type(a)
Out[18]: tuple
```

Example showing how variables make calculations more readable and easier to follow…

```
In [19]: my_income = 100
         tax_rate = 0.1
         my_taxes = my_income * tax_rate

In [20]: my_taxes

Out[20]: 10.0
```