Variables

Common Python Data Types

Data Type	What it stores	Example Value
int	Integers (whole numbers)	8
float	Numbers with decimal places	27.3
str	Sequences of characters	"hello!"

More about variable types: **str**

str is short for String. A variable with the **str** type contains *text*, which is a sequence of zero or more characters. Anything that you can type in on your keyboard counts as a character, so letters, numbers, punctuation, spaces, and more all count.

To create a **str** type variable, quotation marks must be placed around the desired value.

Examples of **str** type values:

```
"Hello!", "123", "b3o;i@$ASI", ""
```

More about variable types: **str**

It is possible to use both single quotes (') and double quotes (") to create a **str** type value.

I would recommend sticking with 1 type, and making it your standard. Swapping back and forth is confusing.

My personal recommendation is **double quotes**, as that is what CodeHS will be using.

More about variable types: **int**

int is short for Integer, which is a term we probably recognize from math. **int** values are always whole numbers - there cannot be decimal places.

int values can be positive, negative or zero.

Examples of **int** type values:

```
5, 0, -100
```

More about variable types: **float**

float is short for Floating-point number. **float** type values, like **int** values, can be positive, negative, or zero. Unlike **int** values, however, **float** values *can* have decimal places.

In fact, any time a **float** value is created, it always has a decimal point, even if there is no value after it.

Examples of **float** type values:

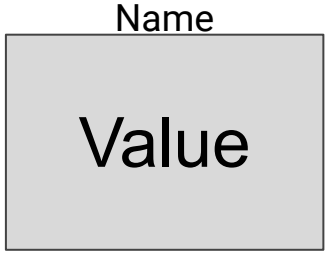
```
12.34, -135.26334363, 0.0
```

All About Variables

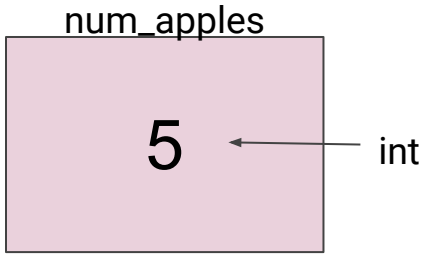
Similarly to in Algebra, a **variable** is a piece of data that has been given a name.

Unlike in Algebra, we'll pretty much always know the value stored within the variables in our programs!

Another way to describe a variable is "a named box which can store a value".



All About Variables



All variables possess 3 qualities:

1. A name
   a. This is how the variable is referenced throughout your program. It should be relevant to the data stored within the variable.
2. A type
   a. This is the Python Data Type that corresponds to the data stored within the variable.
3. A value
   a. This is the actual data stored within the variable.

Naming Variables

1. Use Descriptive Names
   a. Variable names like “a” or “my_variable” don’t *mean* anything - use a name that describes what they are for.
2. Use Underscores Instead of Spaces
   a. Python will wig out and throw an error if you try to use a space in a variable name, so anytime you want to use a space, use an underscore instead.
3. Use Lowercase Letters Only
   a. This one is more of a good rule of conduct than a hard and fast rule, but capital letters are generally saved for a specific purpose.
4. 1st Character Cannot be a Number
   a. Variable names can *include* digits, just not as the first character in their name.
5. No Special Characters
   a. Besides underscores, no special characters are allowed!

Declaring Variables

In order to be able to use a variable in your program, you must first **declare** it.

Declaring a variable means that you're telling the world (a.k.a. your program) that this variable exists!

In Python, a variable is **declared** when you first **assign** it a value! This is also called **initializing** the variable.

Assigning Values

Assigning a value to a variable simply means telling the variable what you want it to remember - putting a value into its box!

This process requires 3 components:

1. The variable's name
2. A single equal sign **=**
3. The value being assigned

```
num_apples = 5
```

Naming Variables

Variables are **case sensitive**. Therefore, the variable names *apples* and *Apples* are different.

```
apples = 8

Apples = 3
```

In the above lines of code, I've **declared** 2 completely separate variables - they aren't connected in any way!

Assigning Values

Once a variable has been **declared**, it can be **assigned** a new value as many times as you like! It will only remember the *most recent* value it was **assigned**, though, and will not remember any previous values that it held!

```
temp = 92.5
temp = 93.7
temp = 96.3
```

`temp` is **initialized** with the value 92.5

`temp` is **assigned** the value 93.7

`temp` is **assigned** the value 96.3

If I access the variable `temp` later in my program, it will **only** tell me 96.3!

Discovering a variable’s type

If you’re running into weird problems in your code, it might be because you are trying to use variables of the wrong type.

In order to determine what type a variable is while code is running, we can call the `type()` function.

Calling the `type()` function on a value will allow you to determine what type of value you’re working with.

Discovering a variable's type

In order for us to **see** the type when we call `type()`, we need to `print()`.

Example:

```
print(type(35))
```

```
<type 'int'>
```

Reading the output

```
<type 'int'>
```

Anytime we call the `type()` function, the output is going to include that word `type` (shocking, I know), followed by the type of the value.

We can also discover the **type** of the value stored inside a variable! We just need to put the variable's name inside the parentheses for `type()`.

String Operators

The 2 String Operators

str values have 2 available operators:

+ and *****.

+ lets us add strings together - this is called **concatenation**. This will only work when both values are **str** type values.
Example: `"what" + "up" = "whatup"`

***** is used with a **str** value on one side, and an **int** value on the other side. This will cause the str value to be repeated.

Example: `"what" * 3 = "whatwhatwhat"`

Using the + operator

One of the most useful places to use the + operator is in the print() function, if we want to print out a specific string at the same time as a variable!

Example:

```
name = "Mr. MacMillan"
print("Hello, " + name)
```

Doing that but with **numbers**!

If we have an int or float typed variable, we cannot simply use the **+** operator, because we'll get an error!

If we're trying to do **MATH**, both sides of the **+** should be numbers, but if we're using words both sides should be **str** values. We can achieve this through typecasting!

Example:

```
age = 5
print("I'm " + str(age) + " years old!")
```