Nonetypes

Quest

So far, we have introduced you to four variable types: ints, floats, booleans, and strings. In this section, we are going to take a very brief look at another special variable type called `None`.

`None` is the keyword that Python uses to indicate when a variable has **no value.**

```
x = None
```

Why would we want a variable to have no value? Well, sometimes when writing a program, we don't know what we want the value of a variable to be at the time when the variable is created. The `None` value can serve as a sort of placeholder for when we need to create a variable before we know what value it needs to have.

Functions Can Also Give the Value None

The example above is not the only way to get a variable to equal `None`. If you recall from the User Input section, we have seen how variables can be set to equal the result of a function call. Well, what happens if the function that we call does not return anything? Do we just get an error?

The answer is no. It is perfectly fine to set a variable equal to the result of a function that returns void. The result is that the variable will store the value `None`:

```
$ Python
>>> x = print("this function returns nothing")
this function returns nothing
>>> x
None
```

Checking for None

`None` is a cool concept to have, but it can cause errors in our code if we are not expecting it. For example, imagine that we have a program that uses a function called `mystery_int()`. `mystery_int()` returns an integer some of the time and `None` otherwise. If `mystery_int()` returns an int, then we want to print the square of it. Don't worry too much about the actual code for `mystery_int()`. We will cover the `return` keyword in the functions section later on.

Look at the code below to see how this might work:

```
import random

def mystery_int():
    """
    return 2 or None with equal probability (0.5)
    you can mostly ignore this code
    """
    if random.random() > 0.5:
        return 2

def main():
    """
    set x equal to mystery_int
    print the square of x
    """

    x = mystery_int()
    print(x)
    print(x * x)

if __name__ == "__main__":
    main()
```

▶ Run   >_ Show

If you run the above code, there's about a 50% chance that it worked and a 50% chance that you got an error. So, if you got an error 🔴 what happened?! Well, `mystery_int()` gave us `None` instead of an integer. Because NoneType is not a number, we cannot use the `*` operator on it. The result is an error. So, how do we fix this? We can use if statements! NoneType is a comparable object which means that we can use the `==` and `!=` comparison operators on it. If you check to see if any actual value == `None`, you will get `False`. Let's update our code.

```
import random

def mystery_int():
    """
    return 2 or None with equal
    probability (0.5)
    """
    if random.random() > 0.5:
        return 2


def main():
    """
    set x equal to mystery_int
    check if x is an integer
    if so, print the square of x
    """

    x = mystery_int()
    print(x)

    if x != None:
        print(x * x)

if __name__ == "__main__":
    main()
```

▶ Run   >_ Show

Yay! Now when `mystery_int()` gives us a value of `None`, we can still have a working program.