Editor

Type Conversion

Section 21.10

Completed

< back to module

Skip straight to editor/terminal

Remember that each variable has its own data *type*. Since Python is *dynamically typed*, the type of the variable depends on the data we assigned to it.

Sometimes you need to convert one type of data to another. For example, consider a case where our program asks the user to input a number, but we want to use this number to create an output that is a string.

If we do:

```
solved_cases = 14

print("Agent S has closed " + solved_cases + " cases this week!")
```

We'll get the following error:

```
Error(s), warning(s):
Traceback (most recent call last):
  File "program.py", line 3, in <module>
    print("Agent S has closed " + solved_cases + " cases this week!")
TypeError: cannot concatenate 'str' and 'int' objects
```

Concatenating only works with strings: we can't concatenate a string and an int together.

In order to get the output we want without triggering an error, we need to modify the variable type using the `str()` function like so:

```
solved_cases = 14
solved_cases_str = str(solved_cases)

print("Agent S has closed " + solved_cases_str + " cases this week!")
```

Running this gives us what we're looking for:

```
Agent S has closed 14 cases this week!
```

And just like we can change an integer to a string, we can also change a string to an integer (as long as the string only contains numbers).

```
morning_coffee = '3'
afternoon_coffee = 2
total_coffee = int(morning_coffee) + afternoon_coffee

print(total_coffee)
```

By using the `int()` function, we changed the type of `morning_coffee` which was a string (note the quote marks) to an integer so that we could add the day's coffee totals together.

You can also convert to a float using the `float()` method:

```
pocket_money = 5
coffee_price = '3.50'
money_left = float(pocket_money) - float(coffee_price)

print(money_left)
```

The above code will turn both the integer `pocket_money` and the string `coffee_price` into floats that we can then do some arithmetic on.

Next, let's look at a more interesting type conversion, an integer to a boolean:

```
flag_found = 1
is_flag_found = bool(flag_found)

print(is_flag_found)
```

Notice this one is a bit different: when we print `is_flag_found`, which has been turned from the integer 1 into a boolean value, we get `True`. If we set `flag_found` to 0, the value of `is_flag_found` when printed would be `False`.

What happens if we use any number *other* than 1 or 0?

In Python, any number that isn't a 0 will convert to a `True` using `bool()`. Even a negative number will convert to `True`.

Additionally, when we use `bool()` on a string value, like so:

```
flag_found = 'yes'
is_flag_found = bool(flag_found)

print(is_flag_found)
```

This will always print `True` except in a couple of notable circumstances. If we make `flag_found = 'false'` or `flag_found = ''` then `bool()` will make these values `False` instead. Give it a try in the editor below.

Time Remaining: 59 mins **Language:** Python 2 | Python 3 | C **Theme:** Dark | Light **Status:** Editor Terminal Server **Server:** Reset **Log:** Download

Steps

Step 1

Try playing around with different types of variables and turning them into other types.

Editor - file: ~/output.py

```
1
```

Save

Terminal - user: fjaxiuQJrZ

```
   ______      __              ______          __
  / ____/_  __/ /_  ___  _____/ ___// /_____ ______/ /_
 / /   / / / / __ \/ _ \/ ___/\__ \/ __/ __ `/ ___/ __/
/ /___/ /_/ / /_/ /  __/ /   ___/ / /_/ /_/ / /  / /_
\____/\__, /_.___/\___/_/   /____/\__/\__,_/_/   \__/
     /____/                    __  _       __
    / ____/_____________  ____  / /_(_)___ _/ /____
   / __/ / ___/ ___/ _ \/ __ \/ __/ / __ `/ / ___/
  / /___(__  |__  )  __/ / / / /_/ / /_/ / (__  )
 /_____/____/____/\___/_/ /_/\__/_/\__,_/_/____/
fjaxiuQJrZ@91be22dd5d1b:~$
```

Completed

< back to module