Previous Next

Unit 1 / Lesson 1 / Project 6

Code From Scratch Project: Fizz Buzz

Estimated Time: 4-5 hours

In this assignment, you'll build a FizzBuzz app from scratch. In the original game of Fizz Buzz, you count from 1 to a given number (let's say 100). If the number is divisible by 3, instead of the number you say "fizz". If it's divisible by 5, you say "buzz". And finally if it's divisible by both 3 and 5, you say "fizz buzz". So, counting from one to fifteen, you'd say: "One, two, fizz, four, buzz, fizz, seven, eight, fizz, buzz, eleven, fizz, thirteen, fourteen, fizz buzz".

FizzBuzz is a great small project to do as it brings together all of the programming fundamentals we've learned so far. And it's also a popular problem that interviewees will be asked to solve in technical interviews. According to some commentators, the ability to code up FizzBuzz will put you amongst the top half percent of programmers who actually know how to program!

Project Requirements

- Have a hard-coded upper line, *n*.
- Print "Fizz buzz counting up to *n*", substituting in the number we'll be counting up to.

- Print out each number from 1 to *n*, replacing with Fizzes and Buzzes as appropriate.
- Print the digits rather than the text representation of the number (i.e. print *13* rather than *thirteen*).
- Each number should be printed on a new line.

Hints and tips

Saving and running code

For this (and future) projects, instead of writing code straight into the interpreter we will be using the Cloud9 editor to save the code to a file. Then we will run the code from the command line. To get started on the project open up a new file (*File* > *New*), then save it as *fizzbuzz.py* (*File* > *Save*, then entering *fizzbuzz.py* into the dialog). To run your code, switch into the console and type `python3 fizzbuzz.py`, then hit enter. You should see any output from your code printed to the console.

Extra Challenge

Create a second version of your FizzBuzz program which has a user-supplied upper limit.

- If the user supplies a value at the command line when script runs, we'll use that value.
- Otherwise, we'll use the `input()` function to get a value from the user.

Hints and tips

Using sys.argv

We can use the `argv` list from Python's built in sys module to allow users to supply values when they run a script from the command line. The best way to see how sys.argv works is with a working example:

```
import sys

print("The name of this script is {}".format(sys.argv[0]))
print("User supplied {} arguments at run time".format(len(sys.argv)))

for arg in sys.argv[1:]:
  print(arg)
```

First we import the `sys` module. This gives us access to the code in `sys`, including the `argv` list. Then we print the number of elements in the `argv` list, and what each of the values are.

Copy the code into a new file called `argv_example.py` and run it. The first time run `python3 argv_example.py`. The second time run `python3 argv_example.py 1 3 cats`.

The first run demonstrates that the first argument to `sys.argv` is always the name of the script that the user has supplied at the command line. It also tells us that there were 0 additional user supplied arguments.

The second time we run it, we see that the first through *n*th arguments get printed, each on a new line.

Note that for the extra challenge your FizzBuzz script should still run even if the user doesn't supply additional command line arguments. This means you'll have to use control flow to deal with two situations: one in which the user supplies command line arguments, and one where they do not.

Using input()

If the user doesn't supply arguments when they call the script, you'll need to use `input()` to get input. All you need to know about `input` is that it allows you to supply a message that prompts the user for a string, and then assigns that input to variable. Here's a very simple example:

```
my_input = input("Enter something, yo!")
print(my_input)
```

As you can see, the value you supply to the raw input gets assigned to a variable. We then print the variable in the second line.

> **Note**
>
> Python 2 vs Python 3: In Python 2 the `input` function is named `raw_input`.

Type conversion

The values you get from both `input()` and `sys.argv` will be strings, and our Fizzbuzz program requires integers. You can convert numbers stored as strings into integers with the `int()` function. For instance `print(type(int('1')))` results in `<class 'int'>`.

Project Completion

When you finish this project, copy it into a Gist and send a link to your mentor. Be sure to discuss and review your code at your next mentor session.

Extension exercise

Earlier in the lesson you learned about exception handling. Practice this skill by writing code that handles users supplying non-numeric inputs to either

sys.argv or input(). First you should figure out where this user behavior would raise an exception in your code, then use a try/except block to catch it. When you catch the error you should print a message telling the user that they need to supply numeric inputs, then use input() to ask for a new value.

☆ ☆ ☆ ☆ ☆ · Report a typo or other issue

https://gist.github.com/ray-newby/522f80ce

Completed

Previous Next