Week 9: Is "Heterological" Heterological?

Response by: **Booth- Alex Li, Cal Hartzell, Evan Rose, Denis Callinan, Jaden Kyler-Wank, Ajay Desai**

Collaborators and Resources: None

Problem 3 *Self-Rejection*

In the *Self-Rejection (An Uncomputable Function)* video, we gave a specific uncomputable function. Below, we begin a similar proof, but in the context of Python code. Help us complete this proof below of a function not computable by Python. (Note: you will find Section 9.3.2 of the TCS book helpful for this also.)

Definition 1 (`self_rejecting_py`) The Python function `self_rejecting_py(w)` should behave as follows for input string `w`:

1. if `w` is anything other than syntactically valid Python source code that defines a function which takes a single input parameter, return `True`.
2. Otherwise (meaning `w` is Python code for a function that takes a single string as input), then return the negation of the output that invoking the program `w` on the input string `w` returns.

(Note that "negation" means what we expect (`NOT`) if the output is a Python Boolean, but is also defined for other outputs, which Python can interpret as Boolean values. It is fine to ignore these typing issues and assume you only need to deal with normal Boolean values.)

With this in mind, we might make the following attempt at implementing `self_rejecting_py`. This function will take the source code, check that it is a python function with one input parameter (using `one_input(w)`), then modifies the source code to execute the function on itself and save the answer to a file (using `add_self_invoke`), runs the modified code (using `exec`, which is essentially Python's universal Turing Machine), then answers the opposite of the file's contents. (Note that all subroutines mentioned for this function can be computed, see us in office hours to discuss how.)

```
    def self_rejecting_py(w):
        if not one_input(w):
            return True
        modified_w = add_self_invoke(w)
        exec(modified_w)
        return not read_result()
```

So for example, running `self_rejecting_py` on the string:

```
def f(m):
  if len(m) % 2 == 0:
    return True
  return False
```

would generate and run the modified code:

```
def f(m):
  return len(m) % 2 == 0
x = '''
def f(m):
  return len(m) % 2 == 0
'''
if f(x):
  print("True", file=open('outputfile'))
else:
  print("False", file=open('outputfile'))
```

What happens if we run the `self_rejecting_py` program on its own source? Show that this function cannot be implemented as described.

Because of the `add_self_invoke` function (which creates code that runs the function described by w and saves that to a file), and the `exec` function, which runs the code created by `add_self_invoke`, running `self_reject_py` would lead to an infinite loop. This is because every call of `self_rejecting_py` will lead to another call of `self_rejecting_py`, as `exec(modified_w)` would start another call of `self_rejecting_py`. Because running this function on itself would cause an infinite loop, the function cannot be implemented as described.

