CS350 – Impossible Python Programs

Purpose: We will blow your mind by looking at some impossible Python programs. The purpose of this module is to show that some problems are impossible by using proof by contradiction – assuming the problem can be computed and showing this leads to a contradiction.

Knowledge: This module will help you become familiar with the following content knowledge:

- Using proof by contradiction to show a problem can not be computed.

Activity: With your group perform the following tasks and answer the questions. You will be reporting your answers back to the class in 60 minutes.

1. This is going to be a real workout for your brain so we will start with some warm-up exercises. Determine whether each of the following statements is either true, false, there is not enough information to decide between true or false, or other.

   (a) Statement: "A decision program outputs either 'yes' or 'no' "

   (b) Statement: "Jill has a hidden piece of paper in her pocket with the number 42 written on it"

   (c) Statement: "This sentence is 35 characters long"

   (d) Statement: "This sentence is in French"

   (e) Statement: "This sentence is false"

   Did that last statement hurt your brain? It is an example of a paradox – a self-contradictory statement that can be neither true nor false.

2. We will use paradoxes to produce impossible programs but we will first look at the following definition.

$$\texttt{yesOnString.py}(P, I) = \begin{cases} \text{“yes”} & \text{if } P \text{ is a Python program, } P(I) \\ & \text{is defined, and } P(I) = \text{“yes”}, \\ \text{“no”} & \text{otherwise.} \end{cases}$$

   (a) What is the value of `yesOnString("not a program","CAGT")`?
   (b) What is the value of `yesOnString(rf('containsGAGA.py'),"CAGT")`?
   (c) What is the value of `yesOnString(rf('containsGAGA.py'),rf('containsGAGA.py'))`?
   (d) What is the value of `yesOnString(rf('yes.py'),rf('containsGAGA.py'))`?
   (e) What is the value of `yesOnString(rf('yes.py'),rf('yes.py'))`?
   (f) What is the value of `yesOnString(rf('longerThan1K.py'),rf('longerThan1K.py'))`?

3. Now look at this defintion:

$$\texttt{yesOnSelf.py}(P) = \begin{cases} \text{“yes”} & \text{if } P \text{ is a Python program, } P(P) \\ & \text{is defined, and } P(P) = \text{“yes”}, \\ \text{“no”} & \text{otherwise.} \end{cases}$$

   (a) What is the value of `yesOnSelf("not a program")`?
   (b) What is the value of `yesOnSelf(rf('containsGAGA.py'))`?
   (c) What is the value of `yesOnSelf(rf('longerThan1K.py'))`?

4. Now look at this definition:

$$\texttt{notYesOnSelf.py}(P) = \begin{cases} \text{“no”} & \text{if } P \text{ is a Python program, } P(P) \\ & \text{is defined, and } P(P) = \text{“yes”}, \\ \text{“yes”} & \text{otherwise.} \end{cases}$$

   (a) What is the value of `notYesOnSelf("not a program")`?
   (b) What is the value of `notYesOnSelf(rf('containsGAGA.py'))`?
   (c) What is the value of `notYesOnSelf(rf('notYesOnSelf.py'))`?

   Ouch! That last one might have hurt the brain. It is another paradox which means that this program is impossible.

5. We can show that yesOnString.py doesn't exist either. Suppose it does exist and we can use an `import` statement to use it in another program like this:

```
from yesOnString import yesOnString
def yesOnSelf(progString):
    return yesOnString(progString, progString)
```

```
def notYesOnSelf(progString):
    val = yesOnSelf(progString)
    if val == 'yes':
        return 'no'
    else:
        return 'yes'
```

Write a sentence or two explaining why this proves that `yesOnString.py` is not computable.

6. We can combine many tricks together into one program to get a much shorter proof by contradiction

```
from yesOnString import yesOnString
def weirdYesOnString(progString):
    if yesOnString(progString, progString)=='yes':
        return 'no'
    else:
        return 'yes'
```

The proof that `yesOnString.py` doesn't exist would be as follows:

1 Assume `yesOnString.py` exists

2 Create `weirdYesOnString.py` as above

3 Explain why `weirdYesOnString` produces a contradiction.

Write a sentence or two which performs the last step in the proof by explaining why you get a contradiction.

7. Similar reasoning shows that no program can correctly predict, for all possible inputs, whether a program will crash or not.

Program name	**Program behavior**
`crashOnString`(P, I)	– return "`yes`", if P crashes on input I – return "`no`", otherwise
↓	
`crashOnSelf`(P)	– return "`yes`", if P crashes on input P – return "`no`", otherwise
↓	
`weirdCrashOnSelf`(P)	– return without crashing, if P crashes on input P – crash, otherwise

Write a three step proof by contradiction proving that `crashOnString.py` does not exist.

Complete the following assignments for grading. Each should be done individually but you may consult with a classmate to discuss strategies.

Assignment 1:
Complete exercises 3.7 and 3.8 on p43 of the text.

Criteria for Success: You have explanations of the output for `noOnSelf` and `yesOnSelf` examples. These are not simple test cases since these two programs are problematic. Therefore your explanations need to address these complications.
Additionally you need an example for when `noOnSelf` and `notYesOnSelf` produce different outputs. Clearly explain why the example produces different results. This requires you to explain why returning 'no' could be different from not returning 'yes'.

Assignment 2:
Complete exercise 3.10 on p43 of the text.

Criteria for Success: You need a complete proof by contradiction. Therefore assume that `definedOnString.py` exists. Then define a new function that uses `defineOnString` and produces a contradiction. This contradiction can be achieved by creating a function that if it is not defined on itself then it would return a legal value and would therefore be defined. Also if it is defined on itself then it would not return a defined value. These paradoxes might make your head hurt but it is a powerful tool.

Assignment 3:
Complete exercise 3.11 on p43 of the text.

Criteria for Success: You need a complete proof by contradiction. Therefore assume that `longerThan10.py` exists. Then define a new function that uses `longerThan10` and produces a contradiction. This will be more fun with paradoxes.

Assignment 4:
Complete exercise 3.12 on p43 of the text.

Criteria for Success: You need a complete proof by contradiction. Therefore assume that `startsWithZ.py` exists. Then define a new function that uses `startsWithZ` and produces a contradiction. Paradoxes rock!

Assignment 5:
Complete exercise 3.14 on p43 of the text. Now that you have done a number of proofs by contradiction you need to see the possible flaws of these arguments.

Criteria for Success: You have a clear explanation of why this proof is not correct.

Assignment 6:
Complete exercise 3.15 on p44 of the text. This question gives a big picture, real life view of what these results might mean or not mean.

Criteria for Success: You have a clear explanation of how these results apply to a real life scenario.

Submit your written answers in Canvas for grading.