Intro CS Final Project Rubric

Overview

This rubric is here to help you understand the expectations for how your project will be evaluated. It is the same rubric that the person evaluating your project will use. You should look at the rubric **before you begin working** on this project **and before you submit it**.

Before you Submit

1. After you've completed the auto-graded question go through each rubric item and do your best to honestly evaluate where you think your project falls.

2. If you think your project "does not meet expectations" for **any** criteria item, you should make any necessary changes.

3. Once you're confident that your project "meets expectations" or "exceeds expectations," you can submit by emailing your code to introCS-project@udacity.com

How Grading Works

1. Your project evaluator will look at the code you submit.

2. Your grade will simply be "pass, meets expectations," "pass, exceeds expectations," or "doesn't pass,"

    a. You earn "pass, meets expectations" if **all** criteria "meet expectations."

b. You earn “pass, exceeds expectations,” if all criteria “exceed expectations” (when possible).
c. Your project “doesn’t pass” if **any** criteria are graded as “doesn’t meet expectations.” In this case, you will have the opportunity to revise and resubmit.

Criteria	Does Not Meet Specifications	Meets Specifications	Exceeds Specifications (Completely Udacious)
Code Functionality			
Code does what it is supposed to do	Code does not pass the autograder when it is Submitted.	Code passes the autograder when it is Submitted.	Code completes the desired task and uses a data structure which allows for efficient execution of the code even with large inputs.
Use of Control Statements			
Control Statements (for, while, if, else) are used appropriately	Selection of control statement is sometimes inappropriate	Selection of control statement is rarely inappropriate	Selection of control statement is never inappropriate.
Looping control statements (for, while) effectively used to avoid repetitive code	Looping control statements are sometimes inappropriately avoided, causing unnecessary repetition in code.	Looping control statements are appropriately used to avoid repetitive code.	*Not Applicable*
Branching control statements (if, elif, else) effectively used to guide code flow.	Branching control statements are used incorrectly or in a way that is unnecessarily complicated.	Branching control statements are used correctly in a way that is not difficult to understand.	Branching control statements are used correctly and logic is expressed concisely and clearly.
Python syntax used to make control statements more intuitive (see Examples below).	Appropriate python syntax is not used at all with control statements, making them difficult to read.	Appropriate python syntax is often used with control statements, making them more intuitive.	*Not Applicable*
Use of Procedures			

Repeated blocks of code are encapsulated into procedures.	Blocks of code are used over and over and not placed in their own procedure	Repetitive blocks of code are contained within their own procedures.	Helper procedures are defined effectively and contribute to code functionality and clarity.
Recursion is used to solve the problem when needed.	Recursion is not used correctly in the problem that asks for it.	Recursion is used correctly in the problem that asks for it.	*Not Applicable*
Arguments and return values are appropriate for procedure purpose.	Choice of arguments and/or return values is not aligned with purpose of procedure.	Choice of arguments and return values is aligned with purpose of procedure.[1]	*Not Applicable*
Make your own Procedure			
MYOP is useful for manipulating and/or analyzing the network.	MYOP is not well planned and designed or does not produce a useful result.	MYOP is well planned and designed, and produces a useful result.	MYOP includes useful results that go beyond the scope of the course. It is obvious that additional research was made in creating the procedure.
Comments are used to explain what MYOP does and, if necessary, how it works.	Comments do not clearly explain MYOP usage and functionality.	Comments clearly explain MYOP usage and functionality.	*Not Applicable*
Code Readability			
Variables and helper procedures are given useful names.	Most variables and helper procedures are not given useful names.	Most variables and helper procedures are given meaningful and useful names.	All variables and helper procedures are given meaningful and useful names.
Variables are used instead of hard-coded values.	Hard-coded values are often used, leading to repetition and lack of clarity.	Appropriate variables are used instead of hard-coded values, clarifying code logic.	*Not Applicable*
Comments are used to explain complex or lengthy segments of code.	Comments are not present in code, or existing comments do not effectively explain code segments.	Comments are present, and effectively explain longer code procedures.	Comments are concise and to the point while still effectively explaining long procedures.
Code is generally neat and ready for personal review.	Code formatting is not ready for personal review.	Code is ready for personal review and neatly formatted.	*Not Applicable*

Examples

Python syntax which makes control statements more intuitive:
To loop through a list or dictionary, you can use:

```
for name in list_names:
  <perform an operation using ‘name’>
```

```
instead of:
```

```
for i in range(0, len(list_names):
    <perform an operation using list_names[i]>
```

To check if an element is in a list/ or if the element is a key in a dictionary:

```
if name in list_names:
  print name + “ is in my list!”
```

You can use empty lists and None values as True/False statements:

```
if list_names is not None:
  print “I know that list_names is not None.”
```

```
if list_names:
  print “I know that list_names is not None and is not an empty list.”
```

[1] For example, a procedure called get_name() should return a name.