Computer Science Foundations

Diagnostic Assessment

This assessment is designed to determine your fit for the Product College's Computer Science Foundations track. Based on your performance on this assessment, you will be placed in either *CS 1: Coding Fundamentals* or *CS 2: Tweet Generator*. Do not think of this as a "test"; this is just our way of ensuring your first course in the CS track is the best fit for your experience level.

***Note:** If you have used Python before, write all code in Python. If not, use a language you know.*

Git and Terminal

Problem 1: Creating a Git Repo

Create a new Git repository with your name to hold all your sample code for this assessment. The repo name should be "`cs-diagnostic-{first name}-{last name}`". For example, Mike Kane's would be titled "`cs-diagnostic-mike-kane`". For any problems that require sample code, name the file as specified in the problem and commit the code to this Git repo.

If you've never used Git before, please put all code into a folder and zip it up when you're done.

Problem 2: Terminal Commands

Create a text file called "`bash_commands.txt`". In this file, please list the corresponding command for each of the following, with each on a separate line:

- Change directory
- List items in current directory
- Print working directory
- Create a new file called "`cookies.txt`"
- Create a new folder called "`recipes`"
- Delete a file
- Stage a file in Git
- Commit a file in Git
- Push the file to the Git remote's origin, on the master branch.

When you are finished, save your work and commit *`terminal_commands.txt`* to the Git repo listed in Problem 1.

Networking and Programming Languages

Problem 3: Web Servers/HTTP

To the best of your ability, explain everything that happens from the moment you type in a web address to the when the website appears in front of the user. In your explanation, try to cover at least some of the following concepts:

- HTTP/S
- DNS
- Packets
- TCP and/or UDP
- Client
- Server
- Headers
- REST

Problem 4: Interpreted and Compiled Languages

What is the difference between an interpreted and a compiled language? What are the strengths and weaknesses of each? Give an example of each, and list a situation for each where a developer should choose one over the other.

Algorithms and Pseudocode

Problem 5: Algorithm Definition

Explain the concept of an algorithm in terms that a young child could understand.

- An algorithm is a set of instructions that you have to do step-by-step to get what you want, like, saaaaay… when you wanna put on your clothes or tie your shoe, you're have to do it a certain way.f

Problem 6: Computational Thinking & Pseudocode

Create a file called `fizzbuzz.py`. At the top of this file, **in a Python comment**, write the **pseudocode** for the following problem:

Write an algorithm (in English) that takes two inputs, "start" and "end" that prints every number from start to end (inclusive). However, if a number is evenly divisible by 3, print the word "fizz" **instead** of the number. If a number is evenly divisible by 5, print "buzz" **instead** of the number. If a number is evenly divisible by both 3 and 5, print "fizzbuzz" **instead** of the number.

Problem 7: Fizzbuzz continued

Under your pseudocode for Problem 6, implement your solution in Python. When you are finished, the file should now contain the pseudocode for the solution (in the form of a Python comment) and the actual code for implementing it. Call the function to prove that it works, and then commit *fizzbuzz.py* to the Git repo mentioned in Problem 1.

Problem 8: Arrays and Collections

In Python, compare and contrast the following collections types:

- List/Array
- Dictionary/Hash Table
- Tuple
- Generator (Extra Credit!)

How are they similar? How are they different?

- List, dictionaries, and tuples can both store a lot of data, but a tuple is immutable meaning that it can't be changed (but can be replaced altogether). Rather than working with indices, a dictionary/hash table has a specific key that corresponds to it.

Problem 9: Arrays and Collections continued

Create a new file titled `collections.py`. In this file, create an example of a List/Array, a Dictionary/Hash Table, and a tuple. Optionally, create an example of a generator, if you like.

Each of these collections should be populated with dummy data. Provide examples of accessing data from each. Make sure you comment your code to make it clear what you're doing!

When you are done with this, commit *collections.py* to the Git repo mentioned in Problem 1.

Problem 10: Functions

Create a new file called `functions.py`. You will use this file for multiple problems, so label the solution to each problem with a comment. (E.g. "# Solution to Problem 11")

A fibonacci number is a number that is generated by the two preceding numbers in the series. Consider the following list of examples:

1 → 2 → 3 → 5 → 8 → 13 → 21 ……...

Create a function `iter_fib` that uses an *iterative* approach to create and print fibonacci numbers. This function should take 1 argument, `num`, which tells the function how many fibonacci numbers to print out. For instance, if the user passes 7 for `num`, the output should be:

1
2

3
5
8
13
21

If the user does not pass a value when calling this function, then `num` should have a default value of 10. When you are done with the function, save the file and commit it to the Git repo created in Problem 1.

Problem 11: Command Line Arguments

Modify the function you created in Problem 11 so that the file is callable from the command line, and can accept arguments as such. For instance, you should be able to pass the argument "7" by calling `$ python functions.py 7` in the terminal.

When you are finished, commit the updated `functions.py` file to Git.

Problem 12: Recursion

Factorials are numbers that are multiplied by every successive number smaller than it. For instance, 3! (pronounced "three factorial") means 3 x 2 x 1 = 6.

Inside *functions.py,* create a recursive function called *recur_factorial* that takes one argument, *num_factorial.* This function should recursively compute the factorial value of the argument passed in.

When you are done, save *functions.py* and commit to the Git repo mentioned in Problem 1.

Object Oriented Programming

Problem 13: Object-Oriented Programming

Create a new file called `OOP_examples.py`. Inside this file, create a class named *Car*. This class should contain the following attributes and methods:

Attributes

- number of wheels
- make
- model
- color
- top speed

Methods

- honk (prints “Beep Beep!” when called)
- description (prints the name of each attribute and the stored value for the object)

Color should be settable by the programmer when the *Car* object is instantiated.

Problem 14: Inheritance

In 1 to 2 paragraphs, explain the concept of Inheritance from OOP and why it is useful to programmers. When possible, use analogies or examples that non-coders can understand.

- Inheritance is convenient to have when you have a superclass that everything has. Why make. If you’re making a bunch of animals, we all know they’re living (mostly), have cells, etc. It’s a good way to handle specificity.
  Are a lot of things going to be using the same function over and over? Best to put it in the superclass instead of writing it all the time.
  Is there something specific only to a certain type of subclass? Like… say, a fish has gills. We don’t want to say that all animals have gills, though fish do. Have it be a subclass.

Problem 15: Abstract Superclasses

Modify your code in `OOP_examples.py` from Problem 14. Create a superclass called *Automobile*, and 3 subclasses that inherit from it:

- Car
- Semi Truck
- Motorcycle

These classes should be semantically accurate to their real-world counterparts--for instance, objects of class *Motorcycle* should have two wheels instead of four. For all three classes, the color attribute should be settable during instantiation.

Reading and Writing to Files

Problem 16: File I/O

Write a sentence in a file named `secret_message.txt` and commit it to your Git repository. Create a file called `read_from_file.py` that can read and print the contents of this file. When you are finished, commit both files to the Git repo mentioned in Problem 1.

Problem 17: File I/O continued

Create a file called `write_to_file.py`. This script should add “{your first name} + {your last name} loves Make School!” to the bottom of `sample_text.txt` when called from the terminal.

When you are finished, run it once to update `sample_text.txt` and then commit both files (`sample_text.txt` and `write_to_file.py`) to the Git repo you set up in Problem 1.

Coding Challenges

Problem 19: Rolling the Dice

Write a function that simulates rolling a 6-sided die by generating a random integer from 1 to 6. Each number should have equal likelihood of being returned. Extra credit: Add an argument to your function, *n*, that lets a programmer choose *n*, the highest number on the *n*-sided die.

Problem 18: Most Common Words

Write a function that takes a string (a sentence or paragraph of text) and a number *k*, then it returns a list of the *k* most commonly occurring words in descending order of frequency.

Problem 20: Linked List Reversal

Reverse a singly linked list. (Use the existing nodes, do *not* create any new node objects).