Homework 1: Doc of Ages

Student 1: Yihan Yang Andrew ID: yihany

Student 2: Nicholas Clark Andrew ID: naclark

Github Repo: https://github.com/YYH1120/17780_hw1_yihany_naclark

Part 1: Find and describe flaws in an APIs documentation

In this part we choose the API is JSON based on Python (https://docs.python.org/3/library/json.html). The JSON API is a traditional API, and its main function is to interchange the format between basic python object and JSON object. In its documentation, we find several flaws as follow:

Flaw 1: Repetitions in sample codes

The JSON API provides some sample codes to teach the user how to use it. However, there are some redundancies in the sample codes. It violates the API guideline of *Documentation should not be needlessly repetitive* ***(D9)***.

```
>>> json.dumps(['foo', {'bar': ('baz', None, 1.0, 2)}])
'["foo", {"bar": ["baz", null, 1.0, 2]}]'
>>> print(json.dumps("\"foo\bar"))
"\"foo\bar"
>>> print(json.dumps('\u1234'))
"\u1234"
>>> print(json.dumps('\\'))
"\\"
>>> print(json.dumps({"c": 0, "b": 0, "a": 0}, sort_keys=True))
{"a": 0, "b": 0, "c": 0}
```

```
>>> mydict = {'4': 5, '6': 7}
>>> json.dumps([1,2,3,mydict], separators=(',', ':'))
'[1,2,3,{"4":5,"6":7}]'
```

```
>>> print(json.dumps({'4': 5, '6': 7}, sort_keys=True, indent=4))
{
    "4": 5,
    "6": 7
}
```

All three pieces of code demonstrate the use of the **dumps** function, which we believe is partially repetitive, such as demonstrating multiple times how to convert data of dictionary type into json format by the **dumps** function. To correct it, I think we should use other types of data that show other data types such as lists and sets as input.

Flaw 2: Missing documentations for some API elements

There is a conversion table which demonstrates the Data type conversion rules between python object and json object. However, it missed out some common python data types such as tuple. This omission can cause problems for users because they have no way to determine the type of the target data format.

JSON	Python
object	dict
array	list
string	str
number (int)	int
number (real)	float
true	True
false	False
null	None

And obviously, it counters the rule of *Method documentation must describe the contract between the method and its caller (**D4**)*. To

complete the contents of contract, we should add other cases of basic format conversion. Check the relevant information to know that tuples will be converted to arrays in JSON object.

Flaw 3: Names of some parameters mislead the users

In the loads function, it contains three following parameters: `parse_float=None`, `parse_int=None`, `parse_constant=None`. We think the names of these parameters would mislead the user because there is an overlap between integer and constant as well as float and constant. In *parse_constant*, it wants to represent some value like positive infinity, negative infinity, or NAN. But the name of parameter would mislead the users. It violates the rule of *Do not mislead or lie to the user (**D6**)*. Actually, we could use *parse_nan* to present some value which is neither integer nor float.

Flaw 4: Documentation are not sufficient to use the API

In the conversion table, it lists eight types of data but in the example codes, it does not demonstrate all the scenarios of format interchanging such as how to convert a boolean type or list type. Therefore, it violates the rule of *Documentation must be sufficient to use the API. It is often appropriate to include high quality code examples (**D7**)*. To correct it, we need to add more examples to demonstrate other scenarios in sample codes.

Flaw 5: Documentation is not organized well

We think there are some problems with the organization of documentations. This document starts by presenting all the sample code to the user. But at this point the user is unfamiliar with the API and cannot quickly read and understand the sample codes. At the same time, the sample code is not focused enough to highlight the explanation of a specific function.

It violates the rule of *Documentation (and the API elements themselves) should be organized in a way that helps communicate the structure of the API, if possible (**D8**)*. We should put the sample codes in a later section and split it up into different examples for using various types of functions.

Flaw 6: Parameter documentation does not say what the parameter represents

In the basic usage of function dump, the meaning of each parameter should be clearly described, as well as the method of usage. However, it does not describe the *cls* parameters and its usage which would cause misunderstanding to users.

```
json.dump(obj, fp, *, skipkeys=False, ensure_ascii=True, check_circular=True,
allow_nan=True, cls=None, indent=None, separators=None, default=None, sort_keys=False,
**kw)
```

Therefore, it violates the rule of *Clearly document all API elements (**D1**)*. To modify the document, it should add the meaning of *cls*. *cls* in Python holds the reference of the class. It is passed as the first argument to

every class methods (methods with @classmethod decorator) by Python itself.

Part 2 Rewrite the documentation

The modified documentation (html file) is in the directory named ‘hw1_part2_yihany_naclark’. And the html file is named ‘hw1_part2_yihany_naclark.html’.

Part 3 Warming up again: Design an API for a thermometer

The compiled files of the documentation is named ‘hw1_part3_yihany_naclark.html’.