MCS 275 — Spring 2022 — David Dumas

Coding Standards

1. Enforcement

To receive full credit, Python code submitted for a graded MCS 275 assignment must follow the rules in this document.

There are several rules, and we know it may take time to learn them. The level of enforcement will be increased over time, with feedback but no point deduction for most infractions of these rules on the first homework.

2. Requirements

(CS1) **Header:** The first few lines of the main Python file of an assignment must be comment lines, containing the following information:
- The course (MCS 275 Spring 2022) and assignment (e.g. Homework 5 or Project 3)
- Your name
- An accurate declaration written in your own words about whether you are the sole author of the work you are submitting, and whether you followed the rules in the syllabus. Alternately, if template code is provided by the instructor, a declaration that the code is based only on the provided template, but you are the sole author of the modifications made to it.

(CS2) **Docstrings:** Every function, module, and class must have a descriptive docstring.

(CS3) **Comments:** Explanatory text is included in the form of comments (starting with `#`), and not in any other way (e.g. string literals used as comments are not acceptable).

(CS4) **Names:** Names of functions, variables, and classes should be descriptive and not abbreviated to the point of incomprehensibility. Iterator variables used in a short loop or comprehension are exceptions, where single character or abbreviated names are permitted.

(CS5) **Indenting:** Each indented code block should be indented by the same amount relative to the surrounding code, and that amount must be at least two spaces (preferred) or one tab (discouraged).

(CS6) **Forbidden construct:** Do not use the construction `range(len(...))` in your code except in the very rare cases when it is the best construction for a task; if you believe you are in the latter situation, confirm with the instructor or TA before proceeding.

(CS7) **No pointless type conversions:** Don't use `str()` on something that is already a string, and don't convert a string to a list just to index it (as strings already support indexing).

That's all. The rest of this document offers clarification and examples, but those are the rules.

3. Requests

These are suggestions, not requirements, but they encourage some good habits and code style.

- Use exactly four spaces for indenting code blocks.
- Use either `"` or `"""` to delimit a string, unless the string itself contains the double quote character. (In general, using one string delimiter consistently is more important than whether you use single or double quote characters.)
- Limit lines of code to a length of 100 characters.

- Avoid the single-letter variable names `l` (lower case letter ell), `O` (upper case letter oh), `I` (upper case letter eye), which are easy to confuse with other symbols in some typefaces.
- Rather than relying on your memory of the rules, before your final submission of any assignment, look through your code with this document open and check each rule in sequence, e.g. read through to check for (CS1), then (CS2), etc.

4. Examples of the requirements

The list of requirements from Section 2 is duplicated here, and each one is illustrated with a code example.

(CS1) **Header:** The first few lines of the main Python file of an assignment must be comment lines, containing the following information:

- The course (MCS 275 Spring 2022) and assignment (e.g. Homework 5 or Project 3)
- Your name
- An accurate declaration written in your own words about whether you are the sole author of the work you are submitting, and whether you followed the rules in the syllabus. Alternately, if template code is provided by the instructor, a declaration that the code is based only on the provided template, but you are the sole author of the modifications made to it.

Here is an example of a compliant header for a Python file written by a student from scratch:

```
# MCS 275 Spring 2022 Homework 38
# Marie Curie
# I am the sole author of this program, and I followed the rules given
# in the assignment and in the course syllabus.
```

And this is an example of a compliant header for an assignment where the course staff provide some kind of template code that students are expected to modify or add to.

```
# MCS 275 Spring 2022 Homework 38
# Emmy Noether
# This program is derived from the template code provided as
# part of the assignment.  I am the sole author of the
# changes to the template code, and I followed the rules in
# the course syllabus and the assignment.
```

(CS2) **Docstrings:** Every function, module, and class must have a descriptive docstring.

- Non-compliant example: This function is not acceptable because it lacks a docstring:

```
def prefix(s):
    return s[:3]
```

- Correction: This modified version is acceptable.

```
def prefix(s):
    """Return the first three characters of a string `s`"""
    return s[:3]
```

(CS3) **Comments:** Explanatory text is included in the form of comments (starting with `#`), and not in any other way (e.g. string literals used as comments are not acceptable).

- Non-compliant example: This loop is not acceptable, because it uses a string literal instead of a comment. (It isn't a docstring, because loops do not have docstrings.)

```
for item in container:
    """Print the weight of each item."""
    print(item.weight)
```

- Correction: This loop contains a comment.

```
for item in container:
    # Print the weight of each item.
    print(item.weight)
```

(CS4) **Names:** Names of functions, variables, and classes should be descriptive and not abbreviated to the point of incomprehensibility. Iterator variables used in a short loop or comprehension are exceptions, where single character or abbreviated names are permitted.

- Non-compliant example: This code uses variable names outside of the list comprehensions that are too short, and which are not descriptive of their purpose.

```
L1 = [ i+1 for i in range(8) ]
Lsp1 = [ x*x+1 for x in L1 ]
```

- Correction: This code is a minimal modification of the example above that is acceptable under the rules.

```
integers = [ i+1 for i in range(8) ]
squares_plus_one = [ x*x+1 for x in integers ]
```

- Correction: This is a more succinct way to do the same thing as the previous examples if the list `integers` is not actually used again.

```
squares_plus_one = [ (i+1)*(i+1)+1 for i in range(8) ]
```

(CS5) **Indenting:** Each indented code block should be indented by the same amount relative to the surrounding code, and that amount must be at least two spaces (preferred) or one tab (discouraged).

- Non-compliant example: This function body is only indented by a single space, and hence is not acceptable.

```
def tsc(x):
 "Topologist's sine curve function"
 return math.sin(1/x)
```

- Correction: Here is a corrected version.

```
def tsc(x):
    "Topologist's sine curve function"
    return math.sin(1/x)
```

- Non-compliant example: This if-else contains two indented blocks that are indented by different amounts.

```
if is_alien_invasion:
  print("Welcome, space wasp overlords!")
else:
      print("Welcome to Lecture 7 of MCS 275.")
```

- Correction: Here is a corrected version.

```
if is_alien_invasion:
    print("Welcome, space wasp overlords!")
else:
    print("Welcome to Lecture 7 of MCS 275.")
```

(CS6) **Forbidden construct:** Do not use the construction `range(len(...))` in your code except in the very rare cases when it is the best construction for a task; if you believe you are in the latter situation, confirm with the instructor or TA before proceeding.

- Non-compliant example: The following method of iterating over elements of a sequence might work, but it uses the forbidden `range(len(...))` construct needlessly, and hence does not follow the course rules.

```
for i in range(len(users)):
    print("Username:",users[i])
```

- Correction: A compliant way to accomplish the same thing as the example above.

```
for username in users:
    print("Username:",username)
```

- Non-compliant example: Here is another example that does not follow the rules.

```
for i in range(len(users)):
    print("User",i,"has username",users[i])
```

- Correction: One way to fix the previous example is:

```
for i,username in enumerate(users):
    print("User",i,"has username",username)
```

(CS7) **No pointless type conversions:** Don't use `str()` on something that is already a string, and don't convert a string to a list just to index it (as strings already support indexing).

- Non-compliant example: This code reads a line of text from the terminal. The return value of `input()` is already a string, however, so applying `str()` to it does nothing.

```
s = str(input("Enter a line of text: "))
```

- Correction: A compliant way to accomplish the same thing as the example above.

```
s = input("Enter a line of text: ")
```

- Non-compliant example: This code prints the characters at positions 5 and 11 in a string, but it converts the string to a list in order to do so. Strings can be indexed directly, so such conversion is not necessary.

```
s = "Python strings support indexing!"
L = list(s)
print(L[5],L[11])
```

- Correction: A compliant way to accomplish the same thing as the example above.

```
s = "Python strings support indexing!"
print(s[5],s[11])
```

5. Note about future changes

Most likely, this document will not change during the course. However, if this document is updated, an announcement will be made on the course web page and in lecture. Rules added or changed whill apply to any assignment that is published after the update to this document appears. See Section 6 for a list of updates (if any).

6. Revision history of this document

- 2022-01-09 Initial publication