"""
Author: Christopher Treadgold
Date: 29/07/2016
"""

PYTHON STYLE GUIDELINES FOR AMAZON WEB SERVICES CMS
_____

FOREWORD: The purpose of this document is to facilitate consistent python code
throughout the AWS CMS project.

In the event that a topic is not covered by this document, refer
instead to the PEP 8 style guide.

GENERAL CONVENTIONS
==========

CLASS NAMES : Class / ClassName (CapWords convention)
FILE NAMES : file.py / file_name.py (No capital letters)
MAX LINE LENGTH : 79 characters (Most editors can show this visually)
FUNCTION NAMES : function / function_name (No capital letters)
QUOTE TYPE : " (Double quotes)
TAB LENGTH : 4
TAB TYPE : Spaces (Most if not all editors have this as an option)
VARIABLE NAMES : variable / variable_name (No capital letters)

FILE HEADER
==========

A file header should be included at the start of each file, directly below the
python interpreter location.

Template (do not indent):

```
"""
# file_name.py
# Author: <Name>
# Date: <Date>
# Edited: DD/MM/YYYY | <Name>
#         N/D        | <Name> (N/D is only for old files where date is
"""                            missing)
```

VARIABLES

Variables should be named with no capital letters with underscores used to split up words as in:

```
variable_name
```

Attempt to keep variable names concise while not resorting to unuseful names such as: a, temp, or variable1.

FUNCTIONS

Functions should follow the same naming conventions as variables. They should contain a short docstring immediately below the function definition as in:

```
def add_two(num):
    """ Takes a number and adds two to it """

    return num + 2
```

WHITE SPACE

White space should only be added where needed and on occassion to improve readability. There should be no whitespace at the end of lines for instance.

A case that is not particularly obvious is keyword argument use. Where unlike variable assignment, there should be no whitespace between the variable name, assignment operator, and value.

Though it can occasionally improve readibility, variable declarations should not be lined up with one another unless they are related to one another and are only 1-2 characters offset.

Inline comments should be seperated from the # symbol by a single space, and just like any other code should not include trailing whitespace.

Avoid situations such as:

```
variable = dict[ "key" ] [ "key_two" ]
class.function( argument )
def function( argument, argument_two )
function(argument) <----Trailing whitespace
function(keyword_arg = value)

variable          = value
long_name_variable = long_value
mid_variable       = mid_value
```

Written correctly:

```
variable = dict["key"]["key_two"]
class.function(argument)
def function(argument, argument_two)
function(argument)<----No trailing whitespace
function(keyword_arg=value)

variable = value
long_name_variable = long_value
mid_variable = mid_value
```

Acceptable variable alignment situation:

```
variable_a   = value_a
variable_ab  = value_ab
variable_abc = value_abc
```

BLANK LINES
==========

Blank lines should be used to seperate function definitions from one another. Use 2 blank lines between each definition, with 1 blank line between the class definition and the first function definition. This is usually __init__.

Blank lines can also be used to seperate logical sections of code, but should be used sparingly. There is no hard and fast rule in terms of what is meant by "sparingly", so add blank lines with discretion.

MULTIPLE LINE STATEMENTS
==========

In order to conform to the 79 character per line limit, statements will frequently have to be split over multiple lines. This is one of the trickiest aspects to writing readable code, so I will to my best to provide tools that make the job easier. All examples in this section use parentheses but apply also when braces and brackets are used.

Firstly, I will describe practices for multi-line function definitions and calls. I'm sure you will have on occasion found yourself writing a line of code like this:

```
    def function(argument1, argument2, argument3, argument4, argument5, argument6,
argument7, argument8)
```

Terrible. Unfortunately, this starts to happen extremely easily indentation level increases. Fortunaly, fixing the problem is quite simple:

```
    def function(argument1, argument3, argument3, argument4, argument5,
                 argument6, argument7, argument8)
```

Much more readable. Now, this solution works great in the case that there is plenty of space after the opening parenthesis, but what if there isn't? Here's an example:

```
    indent_level_one:
        indent_level_two:
            indent_level_three:
                class.disgustingly_long_function_name_why_oh_why(argument1,
                                                                 argument2,
                                                                 argument3,
                                                                 argument3,
                                                                 argument4,
                                                                 argument5,
                                                                 argument6,
                                                                 argument7,
                                                                 argument8)
```

Sends shivers down your spine right? However again there is a simple solution:

```
    indent_level_one:
        indent_level_two:
```

```
        indent_level_three:
            class.disgustingly_long_function_name_why_oh_why(
                argument1, argument2, argument3, argument4, argument5,
                argument6, argument7, argument8
            )
```

Much better. Make sure to note the position of the closing parenthesis. In cases where an argument list goes more than a single line below the beginning of the function definition, the closing parenthesis goes on its on line with its indentation the same as the beginning on the function definition.

For example this is what it would look like if the argument list wasn't so long:

```
    indent_level_one:
        indent_level_two:
            indent_level_three:
                class.disgustingly_long_function_name_why_oh_why(
                    argument1, argument2, argument3, argument4, argument5)
```

As an important note: Unless vertical alignment is being used (as in the first example of this section), no arguments should appear on the first line.

Now onto multi-line strings. Splitting a string over multiple lines is fortunately extremely simple in python:

```
    print(
        "I am a"
        " happy donut"
        " covered in"
        " delicious treats"
        " please don't eat"
        " me oh god I have a family."
    )
```

Which would print:

```
    "I am a happy donut covered in delicious treats please don't eat me oh god
     I have a family." (Minus the newline)
```

The same effect can be achieved using the escape character "\", however this is discouraged in PEP 8. This is what it looks like in case it's necessary:

```
print: "I am a" \
       " happy donut" \
       " covered in" \
       " delicious treats" \
       " please don't eat" \
       " me oh god I have a family."
```

Using parentheses also works to split expressions into multiple lines:

```
print(2
      + 3
      - 5
      % 10
      / 6)
```

This is identical to:

```
print(2 + 3 - 5 % 10 / 6)
```

When using this approach, the breaks should always occur before operands rather than after them.