PYTHON
Wrting Quality Code

STYLING AND FORMATTING

Be consistent
Have the same formatting style among all project's code. Some IDEs allows developers to control code style and inspects different code style violations. IDEs such as pyCharm, Pydev, Emacs or Eric.

Maximum line length
Avoid disjointed statements per line; Its considered to be a bad practise

> "Limit all lines to a maximum of 79 characters" **(PEP 8)**

Indentation (spaces not tabs)
It is preferred to use 4 spaces rather than tabs. Tabs only used when the code is already indented by tabs.

> "Python 3 disallows mixing the use of tabs and spaces for indentation" **(PEP 8)**

Import statement
Put import statements on separate lines and arrange them in the order they are referenced in the code.
this is OK, though *from random import randinit, choice*

White spaces
Use a single white space from the either side of binary (=, +=, -=, and, or, not) and Comparisons (==, <, >, !=, <>, <=, >=, in, not in, is, is not) operators .
Use 2 blank lines before top-level function and class definitions, and 1 blank line before class method definitions

Parentheses
Don't use parentheses with if condition or return statement unless necessary.

Comments
Write comments to indicate and explain what some part of the code does. Remember incorrect comments are worse than no comments.

Docstrings
Include a docstring in the module, function, class and public method as the first statement which provides a concise summary of their purposes.

def foo ():

```
""" this is a
multi line docstring
"""
```

def foo ():

```
""" this is a liner docstring"""
```

BEST PRACTICES

Lists
Use enumerate() when iterating and tracking indexes

for index, value in enumerate (list)

It is a better practice to avoid counter variables and use enumerate when iterating through lists and keep tracking the positions at the same time. enumerate() also gives the option to start from a particular index if you provide it with the index as a second argument.

Use zip() when iterating over 2 or more lists in parallel

for avalue, bvalue, ... in zip(alist, blist,...):

using zip() will facilitate accessing 2 or more lists' values in parallel.

Use list comprehension when adding or filtering values

a = [3, 4, 5]
a = [i + 3 for i in a] adding 3 to all values in the list
b = [i for i in a if i > 4] accessing only values greater than 4 in the list

Dictionaries
Use get() when a value could be unavailable

dictionary.get(key, default=None)

If a key is missing in a dictionary and it is attempted to get its value that will throw an error, KeyError. Therefore, using get() will solve the problem and will return a default value, None, if the key not existed.

Strings
Use format() for formatting strings instead of '+'

Even when the parameters are all strings. However, use your best judgement to decide.
When checking strings' prefixes and suffixes use **startwith()** and **endwith()** respectively.

Naming convention
_single_underscore Protected methods and properties.
class_ add underscore at the end when using reserved words.

CapWords with Classes.
lowercase_underscores with Methods function and variables.
__double_underscore with Private methods

General
Use tuple when swapping or assigning multiple variables

x, y = 10, -10 <<>> x, y = y, x (done in one line)

one thing that should be known is that python evaluates right-side first then left-side when evaluating assignments not expressions, that makes the above example works perfectly.

Truth value test

It is unnecessary to explicitly compare a value to 0, None or True-False; you can just add it to the if statement since None and 0 considered False.
'|' (means or) in the example.

```
x = 0 | None |
False | True
if x:
  #do something
```