Aditya pawar
FYIT-107

PRACTICAL-7

Aditya Pawar
FYIT_107

PEP 8 -- Style Guide for python code

- Introduction –

This document gives coding conventions for the python code comprising the standard library in the main python distribution the companion informational PEP describing style guidelines for the C code in the C implementation of python

PEP8 and PEP257 (Docstring docume. conventions) were adapted from Guido's original python Style guide essay, with some additions from Barry's Style guide.

This style guide evolves over time as additional conventions are identified and past conventions are rendered obsolete by changes in the language itself.

Many projects have their own coding style guide-lines. In this event of any conflicts, such project-specific guides take precedence for that project.

- A foolish consistency is the hobgoblin of little minds.

one of guido's key insights is that code is read much more often than it is written. The guidelines provided here are intended to improve the readability of code & make it consistent across the wide

Aditya pawar
FYIT-107

Aditya Pawar
FYIT-107

```
# Add a comment, which will
provide some distinction in
editors
# supporting syntax
highlighting
if (this_is_one_thing and
    that_is_another_thing):
    # since both conditions are
true, we can probnicate
    do_something()

# Add some extra identation on
the conditional continuation
line.
if (this_is_one_thing
        and
    that_is_another_thing):
        do_something()
```

The closing brace/bracket/parenthesis on multiline constructs may either line up under the first non-whitespace character of the last line of list, as so

```
my_list = [
    1, 2, 3
    4, 5, 6,
    ]
result =
some_function_that_takes_arguments(
    'a', 'b', 'c',
    'd', 'e', 'f',
    )
```

Aditya pawar
FYIT-107

3 Because the code in question predates the introduction of the guideline and there is no other reason to be modifying that code.

4. When the code needs to remain compatible with older versions of python that don't support the feature recommended by the style guide.

Code lay out

Indentation

Use 4 spaces per ~~into~~ indentation well.

continuation lines should align wrapped elements either vertically using python's implicit line joining inside parentheses, brackets and braces, or using a hanging indent. When using a hanging indent the following should be considered; there should be no arguments on the first line and further indentation should be clearly distinguish itself as a continuation line.

correct:

Aligned with opening delimiter
foo =
long_function_name (var_one,
var_two.

Aditya pawar
FYIT-107

Aditya Pawar
FYIT-107

Line break before or after a binary operator

For decades the recommended style was to break after binary operators. But this can hurt readability in two ways: the operators tend to get scattered across different columns on the screen, and each operator is moved away from its operand and onto the previous line. Here, the eye has to do extra work to tell which items are added and which are subtracted.

```
# wrong:
# operators sit far away from
  their operands
income = (gross_wages +
          taxable_interest +
          (dividends -
qualified_dividends) -
          ira_deduction -
          student_loan_interest
```

To solve this readability problem, mathematicians and their publishers follow the opposite convention. Donald Knuth explains the traditional rule in his computers and typesetting series: "Although formulas within a paragraph always break after binary operations and relations, displayed formulas always break before binary operations".

following the tradition from mathematics usually results in more readable code:

Aditya pawar
FYIT-107


```
# correct:
# easy to match operators with operands
  income = (gross_wages
            + taxable_interest
            + (dividends -
  qualified_dividends)
            - ira_deduction
            -
  student_loan_interest)
```

In python code, it is permissible to break before or after a binary operator, as long as the convensation is consistent locally. For new code Knuth's style is suggested.

Blank lines

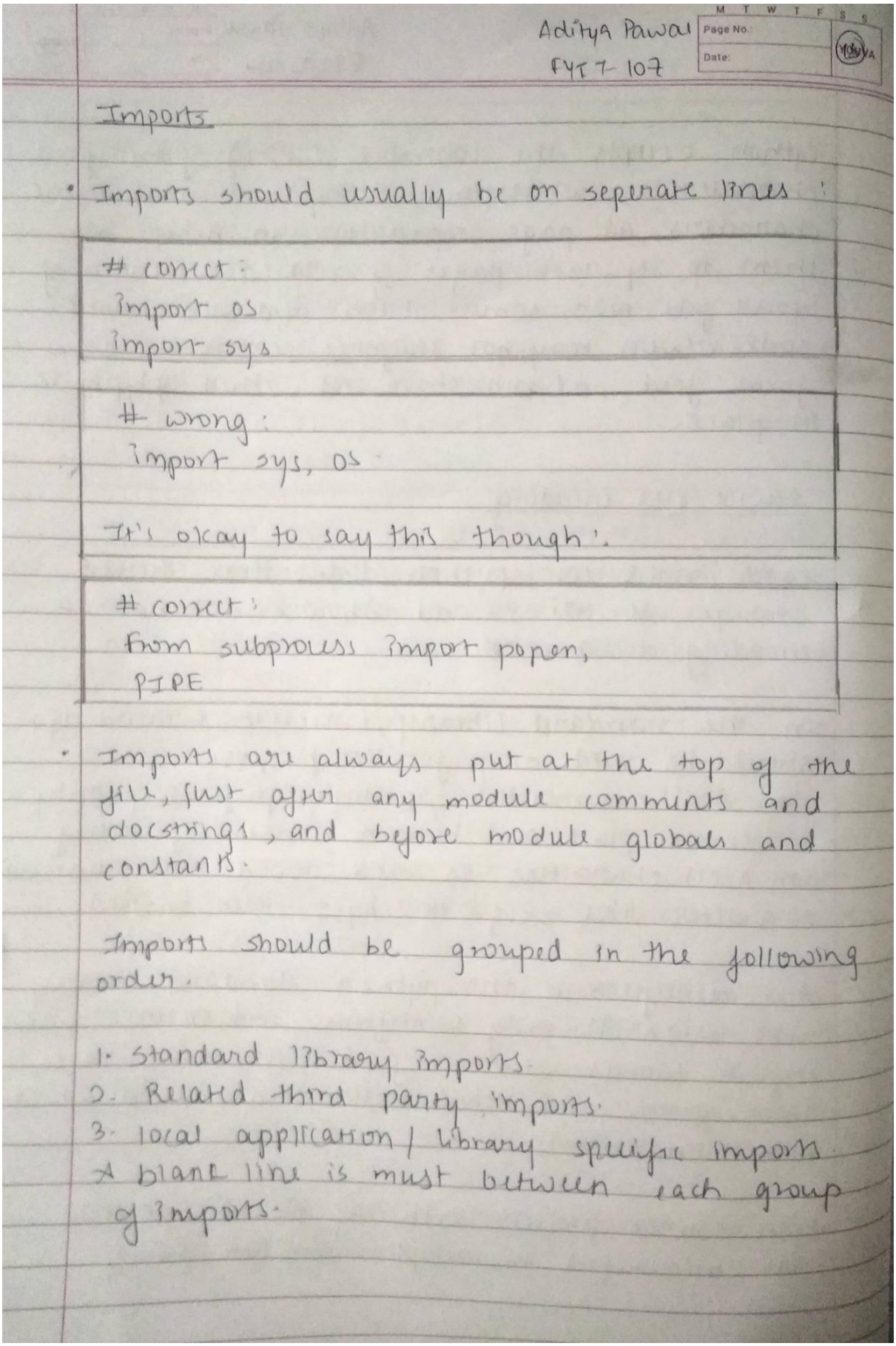
Surround top-level function and class definition with two blank lines.

Method definitions inside a class are surrounded by a single blank line.

Extra blank lines may be used (sparingly) to seperate groups of related functions. blank lines may be omitted between a bunch of related one-liners (e.g. a set of dummy implementations)

Use blank lines in functions, sparingly, to indicate logical sections.

Aditya pawar
FYIT-107



Aditya Pawar
FYIT-107

Page No.:
Date:

Imports

- Imports should usually be on seperate lines :

```
# correct:
import os
import sys
```

```
# wrong:
import sys, os

It's okay to say this though:.
```

```
# correct:
from subprocess import popen,
PIPE
```

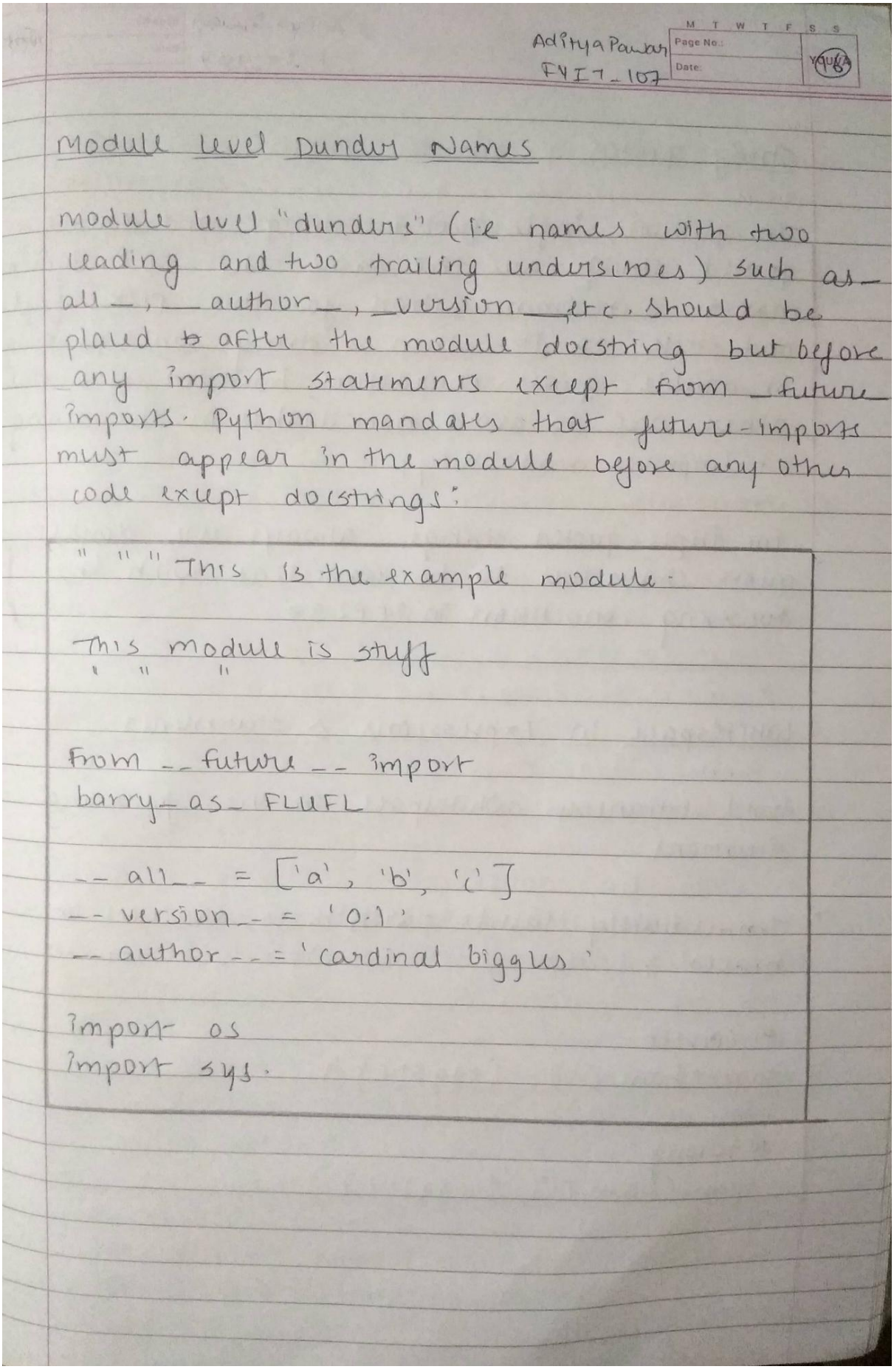
- Imports are always put at the top of the file, just after any module comments and docstrings, and before module globals and constants.

Imports should be grouped in the following order.

1. Standard library imports.
2. Related third party imports.
3. local application / library specific imports.

A blank line is must between each group of imports.

Aditya pawar
FYIT-107



Aditya Pawar
FYIT-107

Module Level Dunder Names

module level "dunders" (i.e names with two leading and two trailing underscores) such as- all__, __author__, __version__ etc. should be placed after the module docstring but before any import statements except from __future__ imports. Python mandates that future-imports must appear in the module before any other code except docstrings:

```
""" This is the example module.

This module is stuff
"""

from __future__ import
barry_as_FLUFL

__all__ = ['a', 'b', 'c']
__version__ = '0.1'
__author__ = 'cardinal biggles'

import os
import sys.
```

Aditya pawar
FYIT-107

Aditya Pawar
FYIT_107

String quotes

In python, single-quoted strings and double-quoted strings are the same. This PEP does not make a recommendation for this pick a rule and stick to it. When a string contains single or double quote characters, however, use the other one to avoid backslashes in the string. It improves readability.

For triple-quoted strings, always use double quote characters to be consistent with the docstring convention in PEP257

Whitespace in Expressions & statements

Avoid extraneous whitespace in the following situations:

- Immediately inside parentheses, brackets or braces:

```
# correct:
spam(ham[1], {eggs: 2})
```

```
# Wrong:
spam( ham[1], { eggs: 2 } )
```

Aditya pawar
FYIT-107

- Between a trailing comma and following close parenthesis:

```
# correct:
foo = (0,)
```

```
# wrong:
bar = (0,)
```

- Immediately before a comma, semicolon, or colon:

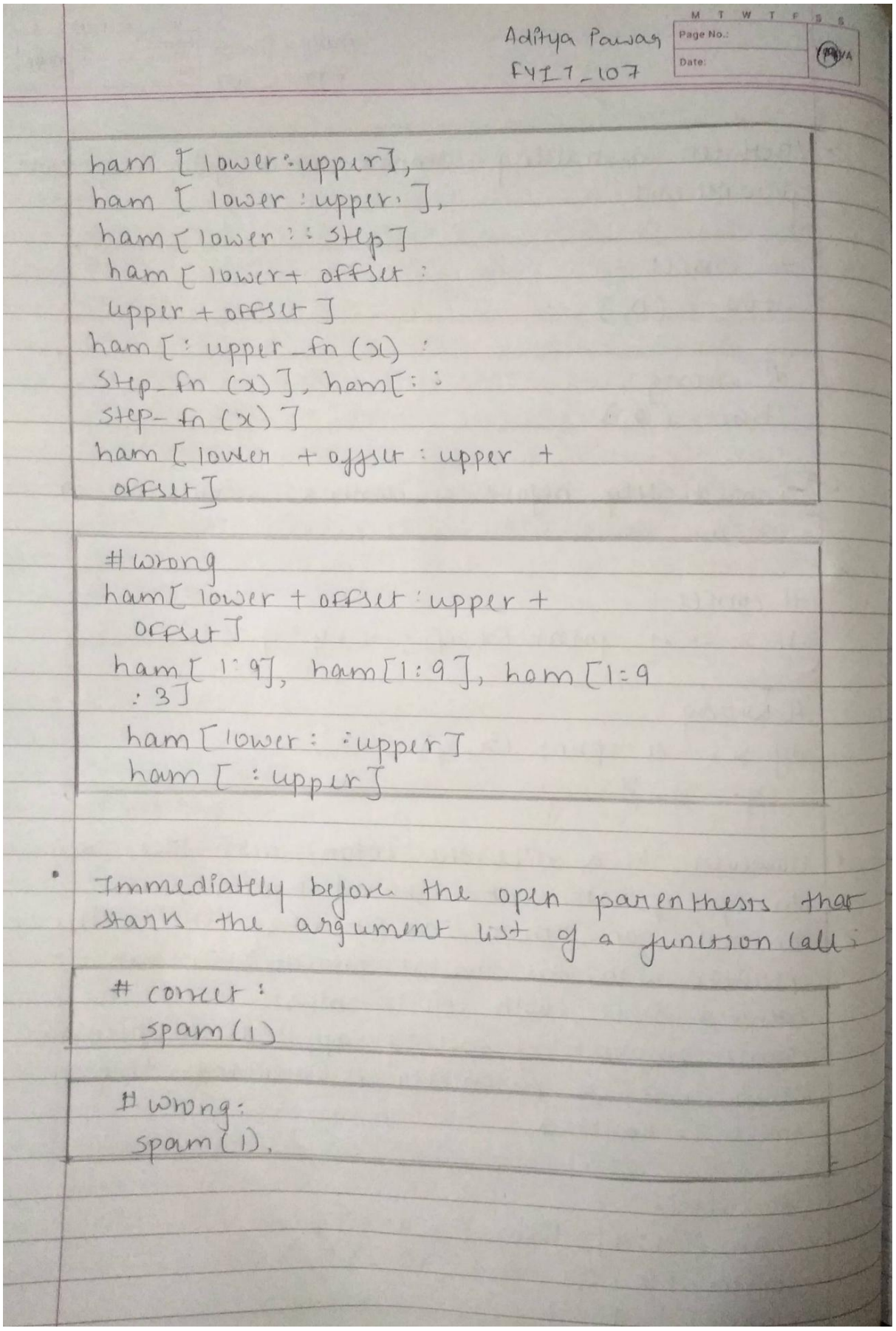
```
# correct:
if x == 4: print(x, y); x, y = y, x
```

```
# wrong:
if x == 4 : print(x, y) ; x,
    y = y, x
```

- However, in a slice the colon acts like a binary operator, and should have equal amounts on either side (treating it as the operator with the lowest priority). In an extended slice, both colons must have the same amount of spacing applied. Exception: when a slice parameter is omitted, the space is omitted:

```
# correct:
ham[1:9], ham[1:9:3],
ham[:9:3], ham[1::3],
ham[1:9:]
```

Aditya pawar
FYIT-107


Aditya Pawar
FYIT_107

```
ham [lower:upper],
ham [ lower : upper: ],
ham [lower : : step ]
 ham [ lower + offset :
 upper + offset ]
ham [ : upper_fn (x) :
step_fn (x) ], ham[ : :
step- fn (x) ]
ham [ lower + offset : upper +
 offset ]
```

```
#wrong
ham[ lower + offset : upper +
  offset ]
ham [ 1 : 9], ham[1:9], ham [1:9
 : 3 ]
 ham [lower : :upper ]
 ham [ : upper ]
```

- Immediately before the open parenthesis that starts the argument list of a function call:

```
# correct :
  spam (1)
```

```
# wrong:
 spam (1).
```

Aditya pawar
FYIT-107

Aditya Pawar
FYIT_107

- Immediately before the open parenthesis that starts an indexing or slicing:

```
# correct:
dct ['key'] = lst [index]
```

```
# wrong:
dct ['key'] = lst [index]
```

- more than one space around an assignment (or other) operator to align it with another:

```
# correct:
x = 1
y = 2
long_variable = 3
```

```
# wrong:
x             = 1
y             = 2
long_variable = 3
```

- Avoid trailing whitespace anywhere because its usualy invisible, it can be confusing: eg: a blackslash followed by space and a newline does not count as a line continuation marker. some editors don't preserve it and many projects (like CPython itself) have pre-commit hooks that reject it.

Aditya pawar
FYIT-107

Aditya Pawar
FYIT_107

Page No.:
Date:

- Always surround these binary operators with a single space on either side: assignment (=), augmented assignment (+=, -= etc.), comparisons (==, <, >, !=, <>, <=, >=, in, not in, is, is not) booleans (and, or, not).

- If operators with different priorities are used consider adding whitespace around the operators with the lowest priority(ies). Use your own judgement; however, never use more than one space, and always have the same amount of whitespace on both sides of binary operators:

```
# correct:
i = 1 + 1
submitted += 1
x = x * 2 - 1
hypot2 = x * x + y * y
c = (a+b) * (a-b)
```

```
# wrong:
i = i+1
submitted += 1
x = x * 2 - 1
hypot2 = x * x + y * y
c = (a+b) * (a-b)
```

- Function annotations should use the normal rules for colons and always have spaces around the -> arrow if present.

Aditya pawar
FYIT-107

Aditya Pawar
FYIT-107

function annotations:

```
# correct:
def munge (input: AnyStr): ...
def munge () -> PosInt: ...
```

```
# wrong:
def munge (input: AnyStr): ...
def munge () -> PosInt: ...
```

Don't use spaces around the = sign when used to indicate a keyword argument, or when used to indicate a default value for an unannotated function parameter:

```
# correct:
def complex (real, imag= 0.0):
return magic (r = real,
i = imag)
```

```
# wrong:
def complex (real, imag = 0.0):
    return magic (r = real, i =
    imag)
```

When combining an argument annotation with a default value, however, do use spaces around the = sign:

```
# correct
def munge (sep: AnyStr = None):
...
def munge (input: AnyStr, sep:
```