Vaishnavi. Nayak
FYIT-54

→ 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: AnyStr =
  None, limit = 1000): ...

# Wrong:
def munge (input : AnyStr = None): ...
def munge (input : AnyStr, limit = 1000):
```

Vaishnavi. Nayak
FYIT - 54

→ Compound statements (multiple statements on the same line) are generally discouraged.

Correct:
```
if foo == 'blah':
    do_blah_thing()
do_one()
do_two()
do_three()
```

Rather not:

Wrong:
```
if foo == 'blah': do_blah_thing()
do_one(); do_two(); do_three()
```

→ While sometimes it's okay to put an if / for / while with a small body on the same line, never do this for multi-clause statements. Also avoid folding such long lines!

Rather not:

Wrong:
```
if foo == 'blah': do_blah_thing()
for x in lst: total += x
while t < 10: t = delay()
```

Vaishnavi. Nayak
FYIT - 54

Definitely not:

Wrong:
if foo == 'blah': do_blah_thing()
else: do_non_blah_thing()
try: something()
finally: cleanup()
do_one(); do_two(); do_three(long,
argument, list, like, this)
if foo == 'blah': one(); two(); three()

* When to Use Trailing Commas
→ Trailing commas are usually optional, except they are mandatory when making a tuple of one element. For clarity, it is recommended to surround the latter in parentheses:

# Correct:
FILES = ('setup.cfg',)

# Wrong:
FILES = 'setup.cfg',

→ When trailing commas are redundant, they are often helpful when a version control system is used, when a list of value, arguments or imported items is expected to be extended over time. The pattern is to

Vaishnavi. Nayak
FYIT-54

put each value on a line by itself, always adding a trailing comma, and add the close parenthesis / bracket / brace on next line. However it does not make sense to have a trailing comma on the same line as closing delimiter.

```
# Correct:
  FILES = [
      'setup.cfg',
      'tox.ini',
      ]
initialize (FILES,
            error = True,
            )

# Wrong:
FILES = ['setup.cfg', 'tox.ini',]
initialize (FILES, error = True,)
```

* Comments

→ Comments that contradict the code are worse than no comments. Always make a priority of keeping the comment up-to-date when the code changes!

→ Comments should be complete sentences. The first word should be capitalized, unless it is an identifier that begins with a lower case letter.

Vaishnavi. Nayak
FYIT-54

→ Block comments generally consist of one or more paragraphs built out of complete sentences. with each sentence ~~of~~ ending in a period.
→ You should use two spaces after a sentence-ending period in multi-sentence comments, except after the final sentence.
→ Ensure that your comments are clear and easily understandable to other speakers of the language you are writing in.
→ Python coders from non-English speaking countries: please write your comments in English, unless you are 120% sure that the code will never be read by people who don't speak your language.

• Block Comments
→ Block Comments generally apply to some code that follows them, and are indented to the same level as that code. Each line of a block comment starts with a # and a single space.
→ Paragraphs inside a block comment are seperated by a line containing a single #.

Vaishnavi Nayak
FYIT-54

- Inline Comments

→ Use inline comments sparingly.

→ An inline comment is a comment on the same line as a statement. Inline comments should be seperated by at least two spaces from the statement. They should start with a # & a single space.

→ Inline comments are unnecessary and in fact distracting if they state the obvious Don't do this:

```
x = x + 1                 # Increment x
```

But sometimes, this is useful:

```
x = x + 1                 # Compensate for border
```