## Imports

```
from __future__ import ... # if needed for P2 compatibility
import standa rd_ package
import third_ par ty_ package as tpp # use abbrev - iation if standard
from x.y import z # z is a module
from x.y import z as t # z collides or too long
```

Don't import individual classes (except *typing* module).
Don't use relative paths.

## __future__

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_ fun ction
```

If you actively need both P2 and P3, use *six*, *future*, *past* instead.

## Naming

```
module_name, package_name, method_name,
function_name, global_var_name, instance_var_name,
function_parameter_name, local_var_name;
_priva te_ var _name;
ClassName, Except ion Name;
GLOBAL _CO NST ANT _NAME.
```

Don't abbreviate too much.
Avoid one-char names for meaningful vars.

## Globals

```
GLOBAL_CONSTANT = "Ok"
_globa l_v ariable = "Try to avoid"
```

Use special functions for external access to global vars.

## Lambda functions

```
operator.mul # ~~lambda x, y: x*y~~
```

For common operators use operator instead.
Ok for one-liners, otherwise use nested funcs.

## Comprehensions

```
simple_case = [i for i in my_list if i > 3]
descri pti ve_name = [
            tra nsf orm ({' key': key, 'value': value}, color= 'bl ack')
           for key, value in genera te_ ite rab - le( som e_i nput)
           if compli cat ed_ con dit ion _is _me - t(key, value)
    ]
```

Each portion must fit on the line and be readable.
Otherwise use a function or loop.

## Conditional expressions

```
simple_case = True if x > 3 else False
the_lo nge st_ ter nar y_s tyl e_t hat _ca n_b - e_done = (
        'yes, true, affirm ative, confirmed, correct'
       if predic ate (value)
       else 'no, false, negative, nay')
```

Each portion must fit on the line and be readable.

## True/False evaluation

```
if seq # ~~if len(seq) > 0~~
# None checks:
x is None
x is not None
```

Use implicit *False* when possible.

## Iterators & Operators

```
for key in my_dict # ~~for key in my_dict.keys()~~
for line in my_file # ~~for line in my_fil e.r ead - lines()~~
```

Use default iterators and operators when supported.

# Google Style for Python Cheat Sheet

by SunnyPhiladelphia via cheatography.com/122856/cs/22993/

## String formatting

```
a + b # only simple cases
'name: %s; score: %d' % (name, n)
'name: {}; score: {}'.fo rma t(name, n)
f'name: {name}; score: {n}' # Python 3.6+
# Long strings:
x = " " "If extra spaces
       are ok"""
x = ("If extra spaces "
               "are not ok")
```

Alternatively, use *textwrap* module.

## Decorators

Only when there's a clear advantage.

Avoid @staticmethod (use a module-level function instead).

Avoid @classmethod (unless you're making a named construtor).

Clearly state that it's a decorator in its docstring.

Avoid external dependencies (files, db connections, etc).

## Properties

Use instead of get-set methods for common private vars.

Use `@property` and `@...setter` decorators.

If access is simple, use public vars instead.

## Exceptions

Custom names should end in `Error`.

Don't catch-all, unless you need to suppress or reraise it.

Minimize the code under `try`.

Use `finally` to clean up.

Use exceptions, not `assert`, to check user-provided args.

## Docstrings and comments

Modules, classes and functions should have docstrings. See full guide for exceptions and details.

Explain tricky bits in comments. Don't literally describe the code.

For inline comments leave 2 spaces before and 1 space after #.

Stick to good English.

## Type annotation

```
def func(a: int) -> List[int]: ...
x = SomeFunc() # type: SomeType
```

Encouraged.

Use *pytype* to type-check at build time.

See more rules in full guide.

## Main

```
def main():
        ...
if __name__ == '__mai n__':
         main()
```

Any file should be importable without unwanted consequences.

## Other

Be consistent with existing codebase.

Use `pylint` or `yapf` for code formatting.

Add shebang to files that are executed.

If function >40 lines, consider splitting it.

Explicitly close files and sockets.

If `with` not possible, use `conte xtl ib.c lo sing()`.

Inherit from `object` explicitly.

Use nested funcs only when needed.

Don't use mutable objects (e.g. lists) as default args.

Instead of `[]` use `None`, and set it to `[]` later.