



# H<sup>0</sup>TMAPS

## Python best practices

---

Prepared by Lucien Zuber

Reviewed by Daniel Hunacek

Date 03.08.2017



Funded by the Horizon 2020 programme  
of the European Union



## PROJECT INFORMATION

---

📍 <b>Project name</b>	Hotmaps
📍 <b>Grant agreement number</b>	723677
📍 <b>Project duration</b>	2016-2020
📍 <b>Project coordinator</b>	Dr. Lukas Kranzl TU Wien - Vienna University of Technology Energy Economics Group – EEG Gusshausstrasse 25-29/370-3 A-1040 Wien / Vienna, Austria Phone: +43 1 58801 370351 E-Mail: <a href="mailto:kranzl@eeg.tuwien.ac.at">kranzl@eeg.tuwien.ac.at</a>

### Legal notice

The sole responsibility for the contents of this publication lies with the authors. It does not necessarily reflect the opinion of the European Union. Neither the INEA nor the European Commission is responsible for any use that may be made of the information contained therein.

All rights reserved; no part of this publication may be translated, reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the written permission of the publisher. Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. The quotation of those designations in whatever way does not imply the conclusion that the use of those designations is legal without the consent of the owner of the trademark.



## **Introduction** **4**

## **Best Practices** **4**

---

1.1 Indentation.....	4
1.2 Organising imports.....	5
1.3 Blank spaces.....	5
1.4 Comments.....	6
1.5 Naming conventions.....	6
1.6 Various recommendations.....	7



# Introduction

---

The purpose of this tutorial is to explain some basics best practices in python. This is based on a documentation that can be found here: <https://www.python.org/dev/peps/pep-0008/#a-foolish-consistency-is-the-hobgoblin-of-little-minds>

## Best Practices

---

### 1.1 Indentation

An indentation level should take 4 spaces.

As indentation is capital in python, it is important to use it correctly. If there is a need to have a command split in several lines, it needs to be done in a clever way:

Yes:

```
# Aligned with opening delimiter.
foo = long_function_name(var_one, var_two,
                          var_three, var_four)

# More indentation included to distinguish this from the rest.
def long_function_name(
    var_one, var_two, var_three,
    var_four):
    print(var_one)

# Hanging indents should add a level.
foo = long_function_name(
    var_one, var_two,
    var_three, var_four)
```

No:

```
# Arguments on first line forbidden when not using vertical
alignment.
foo = long_function_name(var_one, var_two,
                          var_three, var_four)

# Further indentation required as indentation is not distinguishable.
def long_function_name(
    var_one, var_two, var_three,
    var_four):
    print(var_one)
```



## 1.2 Organising imports

The imports should be on several lines

```
Yes: import os
      import sys
```

```
No: import sys, os
```

But you can still say this

```
from subprocess import Popen, PIPE
```

you should group them by:

Standard library imports

Related third party imports

Local application/library specific imports

And put a blank line between those groups

It is better to be the more accurate when you specify a package

```
from mypkg.sibling import example
```

You can also precise them in a relative import

```
from .sibling import example
```

## 1.3 Blank spaces

You should limit the blank spaces to the minimum.

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

```
No: spam( ham[ 1 ], { eggs: 2 } )
```

```
Yes: foo = (0,)
```

```
No: bar = (0, )
```

```
Yes: if x == 4: print x, y; x, y = y, x
```

```
No: if x == 4 : print x , y ; x , y = y , x
```



```
Yes: spam(1)
```

```
No: spam (1)
```

```
Yes:
```

```
x = 1
```

```
y = 2
```

```
No:
```

```
y = 2
```

```
long_variable = 3
```

## 1.4 Comments

It is better to write complete sentences in your comments. It should begin with an uppercase.

Before an inline comment you should enter at least two tabulation

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

sometimes it may be better to have your comments on several lines

## 1.5 Naming conventions

Variables:

They are all in lower-case and are separated by underscore:

```
this_is_a_variable
```

If the variable is a constant, it must be in uppercase and separated by underscore

```
THIS_IS_ANOTHER_VARIABLE
```

If the variable is not meant to be reused by other users (private), it should be written in lower-case and begin with two underscores (for avoiding naming conflict)

```
__this_variable_should_not_be_modified
```

Note that this kind of variable are always accessible by other programmes, but this means that this variable is not supposed to be modified.

Functions:

Functions are supposed to be written in lower case and separated by underscore:

```
this_is_a_function
```

Names to Avoid:

You should not use the characters "l" (l in lowercase), "O" (o in uppercase), "I" (i in uppercase)



#### Modules (file):

They should have short, lowercase name, you can use underscores to improve readability  
module\_name

#### Package:

They should have short, lowercase name, but you shouldn't use underscore, so use only one word

#### Classes:

The name of classes are supposed to be written with CapWords or CamelCase  
ClassFunction

#### Exception:

Since exceptions are like a class, it follows the same rule (CapWords), but you should add Error at the end of it  
NullError

## 1.6 Various recommendations

If you want to compare using None, use *is* or *is not* instead of an operator  
Use *is not* instead of *not ... is*

```
Yes: if foo is not None:  
No:  if not foo is None:
```

Your functions should start with the parameter self

```
Foo(self):
```



[www.hotmaps-project.eu](http://www.hotmaps-project.eu)

**Contact**



Funded by the Horizon 2020 programme  
of the European Union