

5 Effective Python Ways

Jonathan Gustafsson Frennert (14154991)

Follow the PEP 8 Style Guide (Item 2)

Whitespace

```
len(indent) = 4 spaces  
len(line) <= 79 chars  
sep(func, class) = 2 lines  
sep(methods) = 1 line  
{key: value}
```

Naming

Functions, variables, and attributes:
`lowercase_with_underscores`

Protected attributes:
`_single_leading_underscore`

Private attributes:
`__double_leading_underscore`

Classes: `CapitalizedWords`

Constants: `ALL_CAPS`

Expressions

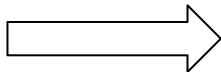
Use inline negation:
`if a is not b`

Use parentheses instead of line continuations (`\`):
`total = (a + b + c + d
 + e + f + g + h)`

Prefer Raising Exceptions to Returning None (Item 20)

```
def careful_divide(a, b):  
    try:  
        return a / b  
    except ZeroDivisionError:  
        return None
```

```
result = careful_divide(1, 0)  
if result is None:  
    print('Invalid inputs')
```

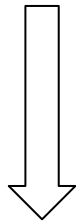


```
def careful_divide(a, b):  
    try:  
        return a / b  
    except ZeroDivisionError:  
        raise ValueError('Invalid inputs')
```

```
try:  
    result = careful_divide(5, 0)  
except ValueError:  
    print('Invalid inputs')  
else:  
    print(f'Result is {result}')
```

Avoid Repeated Work in Comprehensions (Item 29)

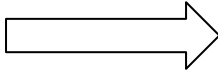
```
found = {name: get_batches(stock.get(name, 0), 8)
         for name in order if get_batches(stock.get(name, 0), 8)}
```



```
found = {name: batches for name in order
         if (batches := get_batches(stock.get(name, 0), 8))}
```

Consider Generators Instead of Returning Lists (Item 30)

```
def index_words(text):  
    result = []  
    if text:  
        result.append(0)  
    for index, letter in enumerate(text):  
        if letter == ' ':  
            result.append(index + 1)  
    return result
```



```
def index_words_iter(text):  
    if text:  
        yield 0  
    for index, letter in enumerate(text):  
        if letter == ' ':  
            yield index + 1
```

Using Packages to Organize Modules (Item 85)

Creation

Add an `__init__.py` file

Project structure:

```
# |---- analysis/  
# |   |---- utils.py  
# |---- frontend/  
# |   |---- utils.py
```

Benefits

Preventing conflicts between modules with identical names:

```
import analysis.utils  
import frontend.utils
```

Manage duplicate names, use **as** to alias imports:

```
from analysis.utils import inspect as analysis_inspect  
from frontend.utils import inspect as frontend_inspect
```

Avoid **import ***:

```
from mypackage.utils import specific_function, another_function
```