Beautiful is better than ugly.

```
if number > 0 and number not in invalid_numbers:
    ...
```

Explicit is better than implicit.

```
class User:
    def __init__(self, name):
        self.name = name

class SecureUser(User):
    def __init__(self, name, password):
        super().__init__(name)
        self.password = password
```

Simple is better than complex.

```
@staticmethod
def method():
    ...
```

Flat is better than nested.

```
def print_items(obj):
    if not hasattr(obj, 'items'):
        return
    for item in obj.items:
        print(item)
```

Readability counts.

```
def reset_password(*users, password=''):
    for user in users:
        user.password = password
```

Errors should never pass silently.

```
>>> def division(a, b):
...     return a / b
...
>>> division(1, 0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in division
ZeroDivisionError: division by zero
```

Unless explicitly silenced.

```
def division(a, b):
    try:
        return a / b
    except ZeroDivisionError:
        return float('nan')
```

In the face of ambiguity, refuse the temptation to guess.

```
>>> a = '5'
>>> a + 1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can´t convert 'int' object to str implicitl
>>> int(a) + 1
6
>>> a + str(1)
'51'
```

There should be one – and preferably only one – obvious way to do it.

```
for i in range(100):
    print(apply_func(i))
```

If the implementation is easy to explain, it may be a good idea.

```
>>> tags = ['<html>', '<body>', '<p>', 'text',
...    '</p>', '</body>', '</html>']
>>> ''.join(tags)
'<html><body><p>text</p></body></html>'
```

Namespaces are one honking great idea – let's do more of those!

```
>>> import math, cmath
>>> math.exp(0)
1.0
>>> cmath.exp(0)
(1+0j)
```

Les règles de style

```
>>> def addition(a : int, b : int) -> int:
...     "Return the sum of numbers `a` and `b`."
...     return a + b
...
>>> help(addition)
```

Don’t repeat yourself (DRY)

```
import sys
import random

def errlog(template, *args):
    print(template.format(*args), file=sys.stderr)

secret = random.randint(0, 100)
guess = int(input('Entrez un nombre de 0 à 100: '))

if guess < secret:
    errlog('Nombre {} trop petit', guess)
elif guess > secret:
    errlog('Nombre {} trop grand', guess)
...
```

We're all consenting adults here

```
class MyObject:
    def __init__(self):
        self._internal = 'internal state'

obj = MyObject()
print(obj._internal)
```

Easier to ask forgiveness than permission (EAFP)

```
try:
    with open('filename', 'r') as f:
        handle_file(f)
except FileNotFoundError as e:
    errlog('Fichier {!r} non trouvé', e.filename)
except PermissionError as e:
    errlog('Fichier {!r} non lisible', e.filename)
```

Unpacking

```
>>> l = [0, (1, 2, {3: 'foo', 4: 'bar'}), 5]
>>> a, (b, c, (d, e)), f = l
>>> print(a, b, c, d, e, f)
0 1 2 3 4 5
>>> x, y, z = 'bar'
>>> print(x, y, z)
b a r
```

Conditions

```
if s:
    print("s n'est pas vide")

name = user.name if user is not None else 'anonymous'
```

Boucle `for`

```
names = ['Alex', 'Alice', 'Bob']
ages = [45, 27, 74]

for name, age in zip(names, ages):
    print(name, age)

for i, (name, age) in enumerate(zip(names, ages)):
    print(i, name, age)
```

Listes en intension

```
squares = [i**2 for i in range(10)]

squares_set = {i**2 for i in range(10)}
squares_dict = {i: i**2 for i in range(10)}
```

Générateurs

```
sum_squares = sum(x**2 for x in range(10))
```

Décorateurs

```
class Circle:
    def __init__(self, cx, cy, radius):
        self.cx, self.cy = cx, cy
        self.radius = radius

    @classmethod
    def from_diameter(cls, ax, ay, bx, by):
        cx, cy = (ax + bx) / 2, (ay + by) / 2
        diam = ((ax - bx)**2 + (ay - by)**2)**0.5
        return cls(cx, cy, diam / 2)

    @property
    def area(self):
        return math.pi * self.radius**2
```

Gestionnaires de contextes

```
with open('hello.txt', 'w') as hello_file:
    print('Hello World!', file=hello_file)
```

Built-in

```
print('{} + {} = {}'.format(2, 3, 2 + 3))

>>> database = {'foo': 123}
>>> database.get('bar')
>>> database.get('bar', 0)
0
>>> database.setdefault('letters', []).append('a')
>>> database.setdefault('letters', []).append('b')
>>> database
{'foo': 123, 'letters': ['a', 'b']}
```

Built-in

```
>>> names = ['Alex', 'Alice', 'Bob']
>>> ages = [45, 27, 74]
>>> list(enumerate(names))
[(0, 'Alex'), (1, 'Alice'), (2, 'Bob')]
>>> dict(zip(names, ages))
{'Alex': 45, 'Alice': 27, 'Bob': 74}
```

collections

```
>>> from collections import Counter
>>> names = ['Alice', 'Bob', 'Bob', 'Alice', 'Alex',
...    'Bob']
>>> count = Counter(names)
>>> count
Counter({'Bob': 3, 'Alice': 2, 'Alex': 1})
>>> count['Alice']
2
>>> count['Camille']
0
```

itertools

```
>>> from itertools import product
>>> for x, y in product(range(10), range(5)):
...     print('{} + {} = {}'.format(x, y, x + y))
...
0 + 0 = 0
0 + 1 = 1
...
9 + 3 = 12
9 + 4 = 13
```

functools,operator

```
>>> import functools, operator
>>> add_3 = functools.partial(operator.add, 3)
>>> add_3(5)
8
```