## Zen of Python

- Objectives
  - Define PEP
  - *Pythonic* code samples

## What Is PEP

- PEP stands for Python Enhancement Proposal
- PEPs intended to be the primary mechanisms for proposing major new features, for collecting community input on an issue, and for documenting the design decisions that have gone into Python
- **PEP Types**
  - A **Standards Track** PEP describes a new feature or implementation for Python
  - An **Informational** PEP describes a Python design issue, or provides general guidelines or information to the Python community
  - A **Process** PEP describes a process surrounding Python such as tools or environment used in Python development

# Style Guide for Python Code – PEP 8

- Python has *official* code guidelines
  - http://www.python.org/dev/peps/pep-0008/
- Code is read much more often than written
  - optimize for readability

**Indentation**:
Use 4 spaces per indentation level.

**Alignment**: with opening delimiter

```
foo = long_function_name(var_one, var_two,
                         var_three, var_four)
```

**Closing brace**: with first character

```
my_list = [
    1, 2, 3,
    4, 5, 6,
]
```

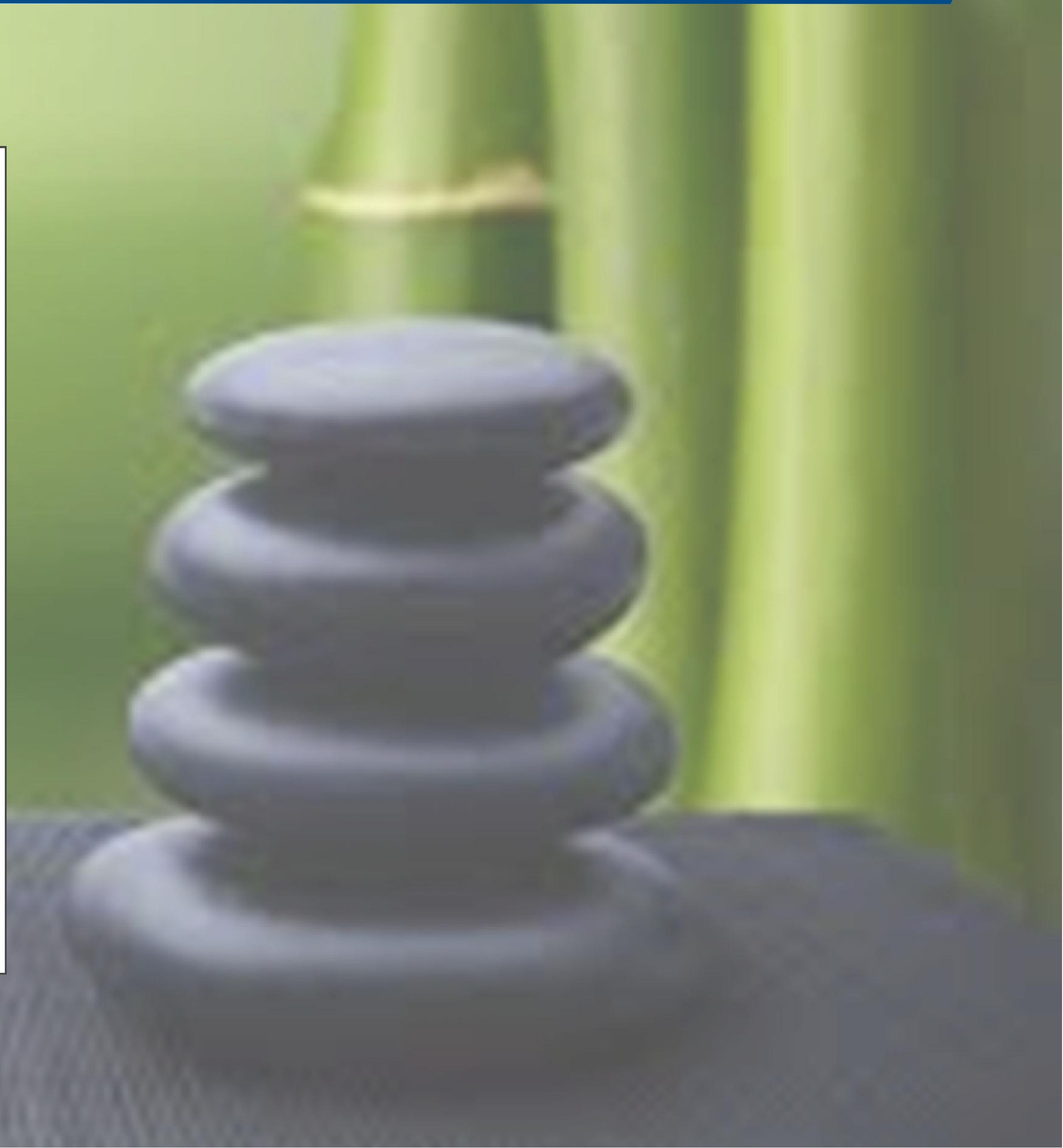
## Style Guide for Python Code – PEP 8

**Tabs or Spaces?**
Spaces are the preferred indentation method.
Python 3 disallows mixing the use of tabs and spaces for indentation

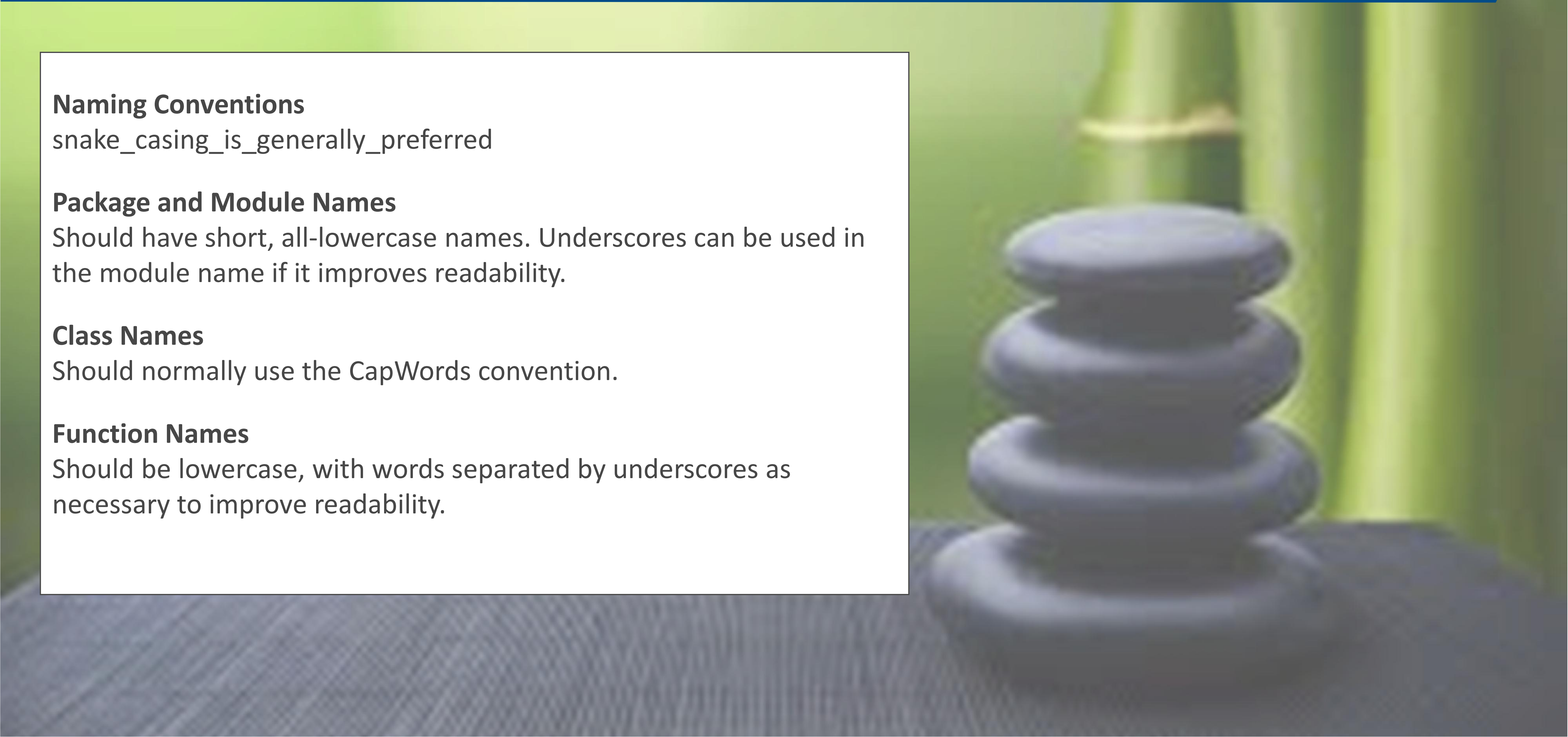
**Blank Lines**
Separate top-level function and class definitions with two blank lines.
Method definitions inside a class are separated by a single blank line.

**Imports**
Imports are always put at the top of the file.
Imports should usually be on separate lines

```
import os
import sys
```



# Style Guide for Python Code – PEP 8



**Naming Conventions**
snake_casing_is_generally_preferred

**Package and Module Names**
Should have short, all-lowercase names. Underscores can be used in the module name if it improves readability.

**Class Names**
Should normally use the CapWords convention.

**Function Names**
Should be lowercase, with words separated by underscores as necessary to improve readability.

## Pythonic code – PEP 20

- Over time, the Python community has settled in on common idioms and styles
  - somewhat motivated by ‘one way to do things’
  - hints at best practices
- Code and patterns that adhere to these idioms are called Pythonic.

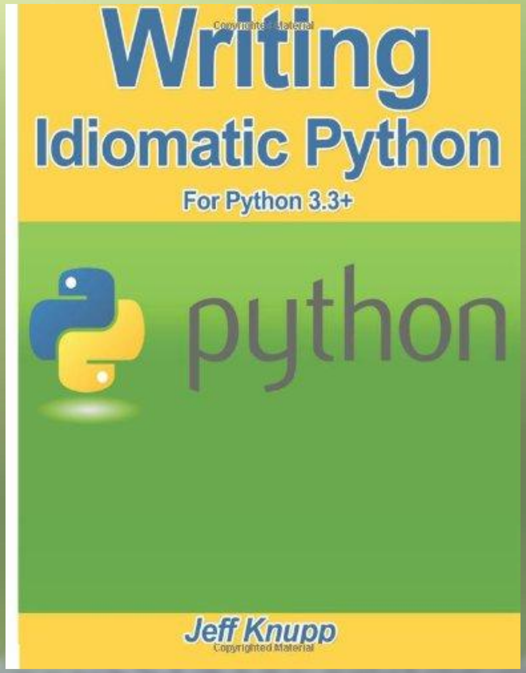
*There should be one -- and preferably only one -- obvious way to do it.*

- Zen of Python

## Pythonic Code Samples

We will explore several areas of code samples around idiomatic code

- Conditionals
- Loops
- Functions
- Exceptions
- Fluent APIs
- Comprehensions
- Collections
- Ecosystem
- Testing



Jeff Knupp at Amazon

Many ideas in this section have been inspired by Jeff's book.

## Pythonic Code: Conditionals

- Avoid placing conditional branch code on the same line as the colon.

**Harmful**

```
user = db.find_user(user_id)
if user.registered: print('welcome back')
```

**Idiomatic**

```
user = db.find_user(user_id)
if user.registered:
    print('welcome back')
```

# Pythonic Code: Conditionals

- Avoid repeating tests on the same variable.

**Harmful**

```
day = get_today()
if day=="Monday" or day=="Wednesday" or day=="Friday":
    print('Today you attend your MWF classes')
```

**Idiomatic**

```
day = get_today()
if day in ("Monday", "Wednesday", "Friday"):
    print('Today you attend your MWF classes')
```

## Pythonic Code: Loops

- Use the enumerate function in loops instead of creating an **index** variable

**Harmful**

```
users = get_new_users()
index = 0
while index < len(users):
    print("Processing {}th user: {}".format(index, users[index].name))
    process_new_user(users[index])
    index += 1
```

**Idiomatic**

```
users = get_new_users()
for index, user in enumerate(users):
    print("Processing {}th user: {}".format(index, user.name))
    process_new_user(user)
```

## Pythonic Code: Exceptions

**Harmful**

```
if user is None:
    return
if not user.email:
    return
if not email_is_well_formed(user.email):
    return
if not mail_server_is_enabled:
    return
# if ... (all conditions)

send_welcome_email(user.email)
```

**Idiomatic**

```
try:
    send_welcome_email(user.email)
except:
    return
```

## Pythonic Code: Fluent APIs

- **Chain string functions** to make a simple series of transformations more clear

**Harmful**

```
txt = user_input.strip()
txt = txt.lower()
txt = txt.split(',')[0]
txt = txt.strip()
```

**Idiomatic**

```
txt = ( user_input
    .strip()
    .lower()
    .split(',')[0]
    .strip() )
```

## Pythonic Code: Comprehensions

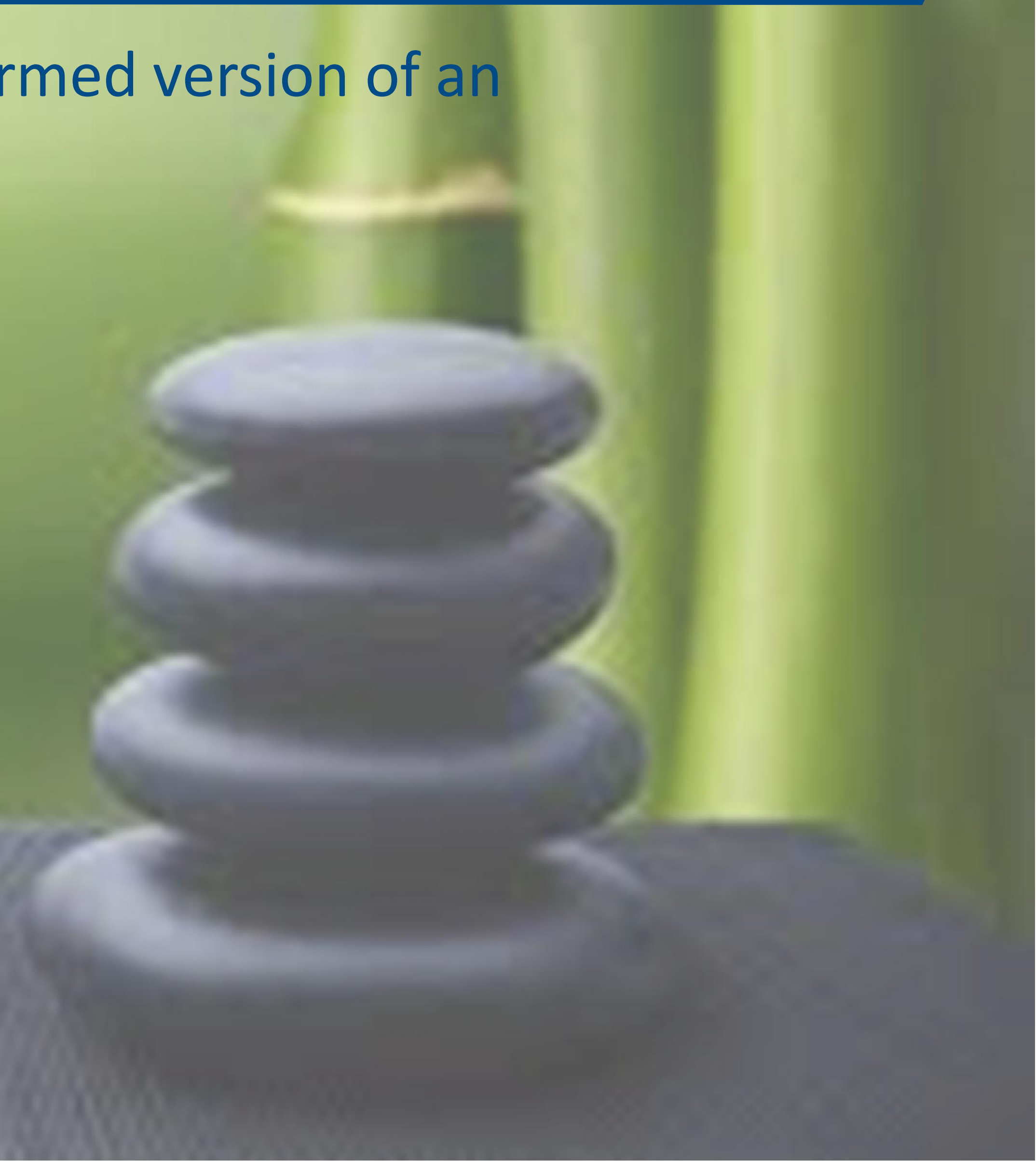
- Use a **list comprehension** to create a transformed version of an existing list.

**Harmful**

```
publishers = []
for b in books:
    if b.is_published:
        publishers.append(b.publisher)
```

**Idiomatic**

```
publishers = [
    b.publisher
    for b in books
    if b.is_published
]
```



# Pythonic Code: Collections

- Use a dictionary as a substitute for a switch statement.

**Harmful**

```
day = 'wed'

if day in ('mon', 'wed', 'fri'):
    do_mwf_schedule()
elif day in ('tues', 'thurs'):
    do_tth_schedule()
elif day in ('sat', 'sun'):
    party()
```

**Idiomatic**

```
actions = {
    "mon": do_mwf_schedule,
    "wed": do_mwf_schedule,
    "fri": do_mwf_schedule,
    "tues": do_tth_schedule,
    "thurs": do_tth_schedule,
    "sat": party,
    "sun": party,
}
actions[day]()
```

Dictionary + function object can stand in for switch / case statements.

# Pythonic Code: Collections

- Use a tuple to **return multiple values** from a function.

**Harmful**

```
def change_multiple(data):
    val1 = 40 # compute...
    val2 = 42 # compute...
    if len(data) == 2:
        data[0] = val1
        data[1] = val2
    else:
        data.append(val1)
        data.append(val2)

lst = []
change_multiple(lst)
val1 = lst[0]
val2 = lst[1]
```

**Idiomatic**

```
def change_multiple():
    val1 = 40 # compute...
    val2 = 42 # compute...
    return val1, val2

val1, val2 = change_multiple()
```

# Pythonic Code: Platform

- **Use sys.exit** in your script to return proper error codes.

**Harmful**

```
def main():
    if not validate_args():
        print('Invalid args...')
        return
```

**Idiomatic**

```
def main():
    if not validate_args():
        print('Invalid args...')
        sys.exit(-5)
        return
```

## Pythonic Code: Ecosystem

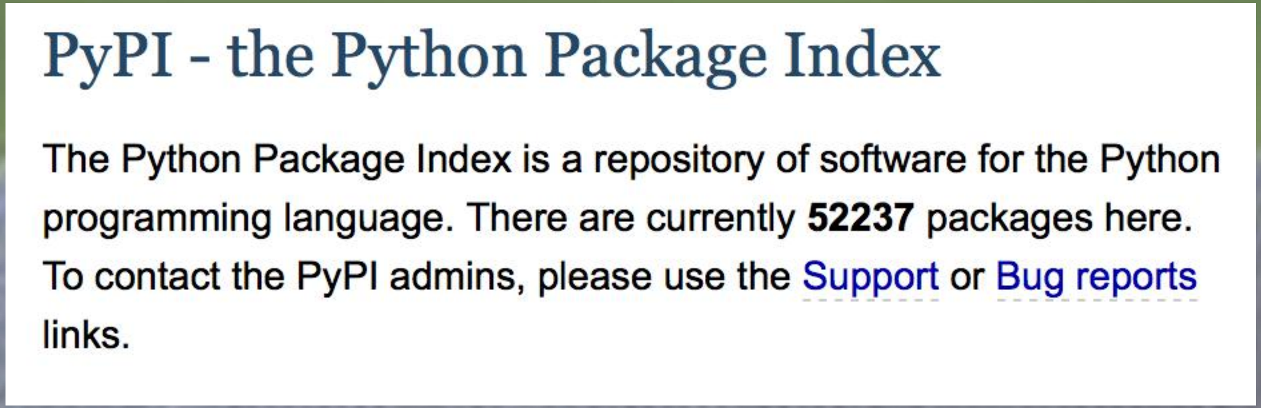
- Avoid reinventing the wheel, get to know PyPI (the Python Package Index)

**Harmful**

```
# let's write that new security component
```

**Idiomatic**

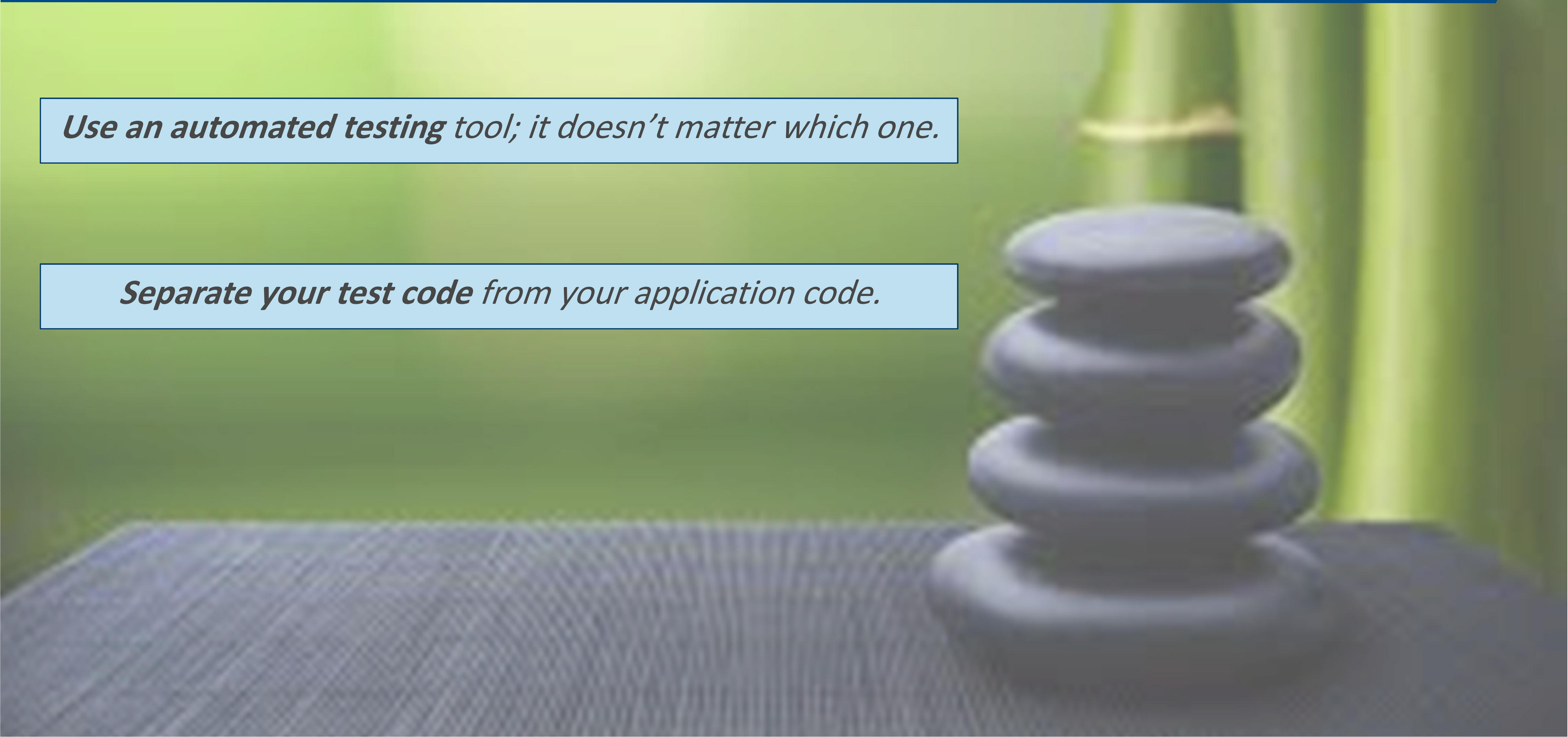
```
pip install security_layer
```



PyPI - the Python Package Index

The Python Package Index is a repository of software for the Python programming language. There are currently **52237** packages here. To contact the PyPI admins, please use the Support or Bug reports links.

# Pythonic Code: Testing



***Use an automated testing** tool; it doesn’t matter which one.*

***Separate your test code** from your application code.*

## Summary

- *Pythonic* code confirms to common idioms used in the community
- C-style algorithms are typically not Pythonic
- PEP 8 has many guidelines for code style