Pythonic / Idiomatic Python

Global Knowledge®

Objectives

- Define *Pythonic* code
- See examples of unpythonic code
- Explore some of Python's coding guidelines
- Use tools to automatically discover issues with code

Pythonic code

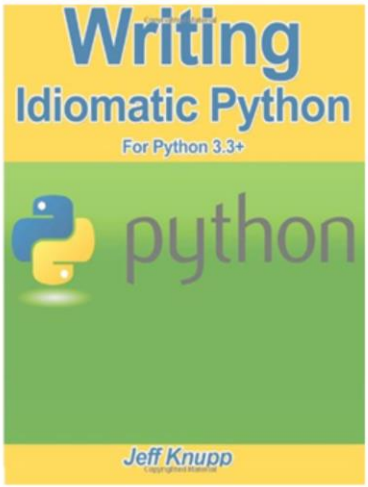
- Over time, the Python community has settled in on common idioms and styles
  - somewhat motivated by 'one way to do things'
  - hints at best practices
- Code and patterns that adhere to these idioms are call *Pythonic*. [1]

> *There should be one -- and preferably only one -- obvious way to do it.*
>
> - Zen of Python

Pythonic vs. Non-Pythonic Code

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: Conditionals

Avoid comparing directly to True, False, or None.

Harmful

```
charge = get_should_charge_card(user)
if charge == True:
    bill_card(total)
```

Would crash if we wrote:
`def get_should(user): return user.card_number`

Idiomatic

```
charge = get_should_charge_card(user)
if charge:
    bill_card(total)
```

Pythonic Code: Conditionals

*Use **is** for comparisons with singletons.*

Harmful

```
def append_data(d, lst = None):
    if not lst:
        lst = []
    lst.append(d)
```

Would not behave correctly with this code:
`lst = []; append_data(42, lst)`

Idiomatic

```
def append_data(d, lst = None):
    if lst is None:
        lst = []
    lst.append(d)
```

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])
```

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: Functions

*Avoid using a **mutable** object as the default value.*

Harmful

```
def append_data(d, lst = []):
    lst.append(d)
    return lst
```

Idiomatic

```
def append_data(d, lst = None):
    if lst is None:
        lst = []
    lst.append(d)
    return lst
```

Why? →

Pythonic Code: Functions

*Avoid using a **mutable** object as the default value.*

Harmful

```
def append_data(d, lst = []):
    lst.append(d)
    return lst
```

```
print(append_data(1))
print(append_data(2))
print(append_data(3))

# Prints (unexpected):
[1]
[1, 2]
[1, 2, 3]
```

Idiomatic

```
def append_data(d, lst = None):
    if lst is None:
        lst = []
    lst.append(d)
    return lst
```

```
print(append_data(1))
print(append_data(2))
print(append_data(3))

# Prints (expected):
[1]
[2]
[3]
```

Pythonic Code: Exceptions

*Use Exceptions to Write Code in an **"EAFP" Style**.*

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
```

EAFP (Easier to Ask for Forgiveness than Permission) vs. LBYL (Look Before You Leap)

EAFP is idiomatic in Python and other languages
LBYL is idiomatic in languages which do not have strong exception support (e.g. C). LBYL is error prone because you can always forget a case (e.g. what if the network is offline?)

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
]
```

Note: There may also be performance benefits to using a list comprehension (or generator expression) due to optimizations in the CPython interpreter.

Pythonic Code: Collections

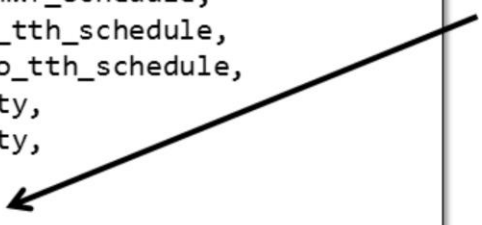
*Use a dictionary as a **substitute for a switch** statement.*

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 **default value** in dict.get to simplify code.*

Harmful

```
log_level = 'normal'
if 'logging' in configuration:
    log_level = configuration['logging']
```

Idiomatic

```
log_level = configuration.get('logging', 'normal')
```

Pythonic Code: Collections

*Use **collections.namedtuple** to make tuple-heavy code more clear.*

Harmful

```
def get_data():
    for row in db.get_books('1st publishing house'):
        yield row

for br in get_data():
    print("{} with isbn {} was published {}".format(
        br[1], br[1], br[2]))
```

How do you know what order to use?
What if the elements change?

Pythonic Code: Collections

*Use **collections.namedtuple** to make tuple-heavy code more clear.*

Idiomatic

```
from collections import namedtuple
BookRow = namedtuple('BookRow', ['isbn', 'title', 'published'])

def get_data():
    for row in db.get_books('1st publishing house'):
        yield BookRow._make(row)

for br in get_data():
    print("{} with isbn {} was published {}".format(
        br.title, br.isbn, br.published))
```

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]
```

Pythonic Code: Collections

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

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: Platform

*Use the **os.path** module when working with directory paths.*

Harmful

```
dir  = "/user/local/bin/"
file = "log.txt"
fullname = dir.rstrip('/') + '/' + file
```

Idiomatic

```
import os

dir  = "/user/local/bin/"
file = "log.txt"

fullname = os.path.join(dir, file)
```

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.*

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

Maximum Line Length
Limit all lines to a maximum of 79 characters.

Line wrapping
Use Python's implied line continuation inside parentheses, brackets and braces.

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 e.g.

```
import os
import sys
```

Style Guide for Python Code – PEP 8

Comments
Comments that contradict the code are worse than no comments.

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.

Comparisons
Comparisons to singletons like `None` should always be done with `is` or `is not`, never the equality operators.

Style Guide for Python Code – PEP 8

Conditional sequences
Use the fact that empty sequences are false. e.g.

```
Yes: if seq:
     if not seq:

No: if len(seq)
    if not len(seq)
```

PyLint - Style and bug detection

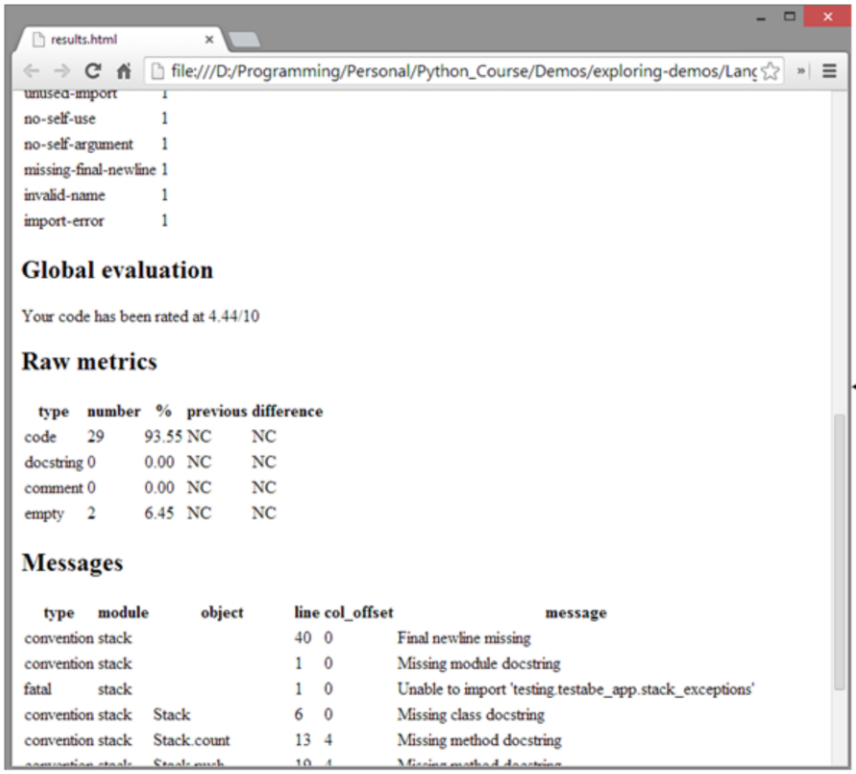
- PyLint is an external package which checks for
  - PEP 8 violations
  - many bugs [details]
  - reports (lines of code, etc)
  - code duplication
  - statistics by type
  - install via '**pip install pylint**'
  - http://www.pylint.org/

PyLint [command line]

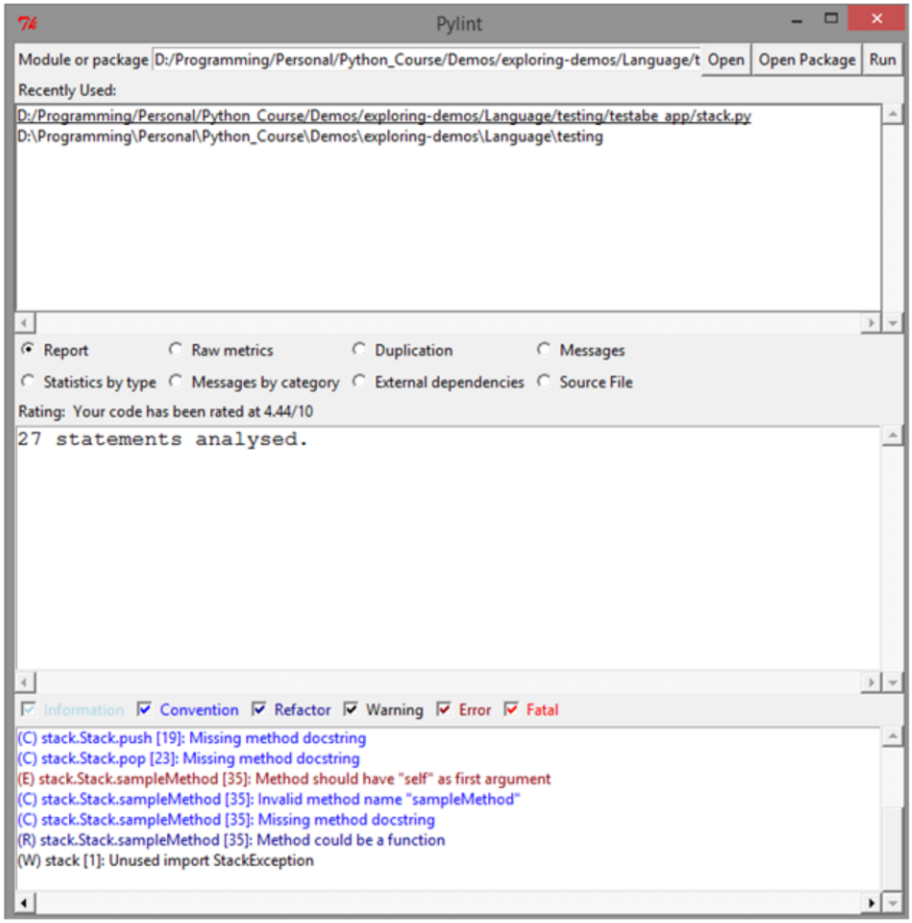
```
D:\dev> pylint stack.py

************* Module stack
C: 40, 0: Final newline missing (missing-final-newline)
C:  1, 0: Missing module docstring (missing-docstring)
C:  6, 0: Missing class docstring (missing-docstring) C: 35, 4:
Invalid method name "sampleMethod" (invalid-name)
C: 35, 4: Missing method docstring (missing-docstring)
E: 35, 4: Method should have "self" as first argument (no-self-
argument)
R: 35, 4: Method could be a function (no-self-use)
W:  1, 0: Unused import StackException (unused-import)


Report
======
27 statements analysed.

Raw metrics ...
```

PyLint [HTML Reports]

```
D:\dev> pylint stack.py -f html > results.html
```



PyLint [GUI]



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
- PyLint automatically uncovers issues with code