Conf42: Python 2021

Ammara Laeeq

Outline

- What is Pythonic and why is important?
- What is code quality?
- Why code quality is important?
- What tools are available?
- What is Idiomatic Python?
- What are some common Pythonic Idioms?

What is Pythonic?

“Any program, function or block of code that follows style guidelines and make use of Python’s unique capabilities or language features”

What is Code Quality?

- It does what it is supposed to do
- There are no defects and problems
- It is readable, maintainable and extendable

Why Code Quality is Important?

- Clarity
- Efficiency
- Credibility

Tools

Zen of Python
PEP8
Linters

Zen of Python (PEP-20)

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

- **May 2020, Barry Warsaw sang these as lyrics**

PEP 8

“Proposals are documents to describe new features proposed and to document different aspects of Python like performance, design and style, for the community”

Code Layout

- One statement per line
- Single point of return
- Indentation
  - 4 spaces per indentation level
- Tabs or Spaces
  - Spaces are preferred
- Maximum Line Length
  - Limit lines to 79 characters
  - 72 characters for comments and docstrings
- Imports
  - Top of the file
  - Each import at a separate line
  - Wild card imports are discouraged
- String Quotes
  - Single or double, be consistent

Naming Conventions

- **Packages/Modules**
  - Short names, all lowercase, _ can be used for readability but are discouraged e.g. pandas, models.py
- **Classes**
  - Use PascalCase, InitCaps, do not include the world Class e.g. Employee not EmployeeClass
- **Exception Names**
  - Class naming rules, using suffix “Error” is recommended e.g. LimitNotExpiredError
- **Constants**
  - Use all caps SNAKE_CASE e.g. DATABASE_URL
- **Functions/Variables**
  - Use standard snake_case e.g. def final_score(), team_score
- **Function/Method Arguments**
  - Self(first to instance methods), cls(first to class methods), a trailing _ to avoid clashes with keywords(class_ is better than clss)
- **Methods and Instance Variables**
  - Function naming rules, 1 leading _ for non public, 2 leading __ to avoid parent/child clashes

Linters

- What is a Linter?

- Logical Lint
  - Code errors
  - Dangerous code patterns
  - Code with potentially unintended results
- Stylistic Lint
  - Code not conforming to conventions and style guides

Linters for Python

- IDEs
  - Mostly stylistic
- Pylint
  - Logical and Stylistic
- pycodestyle
  - Stylistic
- pydocstyle
  - Stylistic
- PyFlakes
  - Logical

When to Check?

- As you write it
- When it’s checked in
- When running your tests

Idiomatic Python

“An idiom is a phrase that doesn't make literal sense, but makes sense once you're acquainted with the culture in which it arose.”

For programming?

“Things you do daily for development in a particular language that are familiar and meaningful to those who work in same language”

Common Python Idioms

- Swap variables without using **temp** variable
  - a, b = b, a
- Do not compare directly to singletons like **True**, **False**, **None** or **0**
- Non Idiomatic: if foo == True:
- Idiomatic: if foo: or if foo is None:
- Use ‘is not’ instead of ‘not … is’
- Non Idiomatic: if not foo is None:
- Idiomatic: if foo is not None:
- Empty sequences are also **False.** This includes [ ], { }, ( )

Repeating Variable Name in if Statement

Non Idiomatic:

```
is_generic_name = False
name = 'Tom'
if name == 'Tom' or name == 'Dick' or name == 'Harry':
    is_generic_name = True
```

Idiomatic:

```
name = 'Tom'
is_generic_name = name in ('Tom', 'Dick', 'Harry')
```

For Loops

Other programming languages:

```
for (int i=0; i < container.size(); ++i)
{
    // Do stuff
}
```

Python:

```
for i in range(len(my_list)):
    print(my_list[i])
```

Idiomatic Python: Use the ‘in’ operator

```
for element in my_list:
    print (element)
```

Looping Backwards

Other programming languages:

```
for (int i=container.size()-1; i>=0; i--)
{
    //Do something
}
```

Python:

```
for i in range(len(my_list)-1, -1, -1):
    print(my_list[i])
```

Idiomatic Python: Use 'reversed'

```
for element in reversed(my_list):
    print(element)
```

For Loops

Tracking Index of for loop

Non Idiomatic:

```
my_container = ['Larry', 'Moe', 'Curly']
index = 0
for element in my_container:
    print ('{} {}'.format(index, element))
    index += 1
```

Idiomatic: Use ‘enumerate’

```
my_container = ['Larry', 'Moe', 'Curly']
for index, element in enumerate(my_container):
    print (f'{index} {element}')
```

For Loops

Using for … else syntax

Non Idiomatic:

```
has_address = False
for user in users:
    if user.address:
        has_address = True
        break
if has_address:
    print('Atleast one user has entered address.')
```

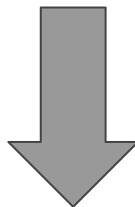
Idiomatic: Use ‘else’

```
for user in users:
    if user.address:
        print('Atleast one user has entered address.')
        break
    else:
        print('Nobody has entered address.')
```

Using 'iter' to Check for Sentinel Value:

```
chunks = []
while True:
    chunk = f.read(32)
    if chunk == '':
        break
    chunks.append(chunk)
```

Better Way



```
chunks = []
for chunk in iter(partial(f.read, 32), ''):
    chunks.append(chunk)
```

Strings

- Use string methods instead of the string module
  - String methods are faster
- Use ‘’.startswith() and ‘’.endswith() instead of slicing for prefix or suffix checking
  - Cleaner and less prone to error
- Use ‘’.join() while creating string from list elements instead of using the ‘+’ operator

Non Idiomatic:

```
result_list = ['True', 'False', 'File not found']
result_string = ''
for result in result_list:
    result_string += result
```

Idiomatic:

```
result_list = ['True', 'False', 'File not found']
result_string = ''.join(result_list)
```

Context Managers

- Objects to be used with the **with** statement
- Make resource management more explicit and safer
- Separates business logic from administrative logic

Non Idiomatic:

```
f = open('file.txt')
a = f.readline()
f.close()
```

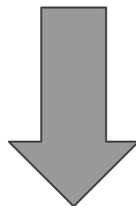
Idiomatic:

```
with open('file.txt') as f:
    for line in f:
        print line
```

Context Managers - Examples

```python
lock = threading.lock()
lock.acquire()
try:
    print('Executing critical section')
finally:
    lock.release()
```

Better Way

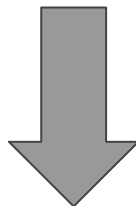


```python
lock = threading.lock()
with lock:
    print('Executing critical section')
```

Context Managers - Examples

```
try:
    os.remove('somefile.txt')
except OSError:
    pass
```

Better Way



```
with ignored(OSError):
    os.remove('somefile.txt')
```

Context Managers - Examples

- Changing the standard output to a file
  - Use redirect_stdout(file) context manager
- Modifying some variable in temporary context
  - Use localcontext(Context) context manager

Lists

- Creating a length-N list of the same thing
  - Use the python list ‘*’ operator to create simple and nested lists

Idiomatic Way:

```
>>> list = ['hello']*5

>>> list

['hello', 'hello', 'hello', 'hello', 'hello']
>>> num_list = [[1,2]]*5

>>> num_list

[[1, 2], [1, 2], [1, 2], [1, 2], [1, 2]]
>>>
```

Lists - Unpacking

- Use the '*' operator to represent the rest of the list instead of slicing

Non Idiomatic:

```
some_list = ['a', 'b', 'c', 'd', 'e']
(first, second, rest) = some_list[0], some_list[1], some_list[2:]
print(rest)
(first, middle, last) = some_list[0], some_list[1:-1], some_list[-1]
print(middle)
(head, penultimate, last) = some_list[:-2], some_list[-2], some_list[-1]
print(head)
```

Idiomatic:

```
some_list = ['a', 'b', 'c', 'd', 'e']
(first, second, *rest) = some_list
print(rest)
(first, *middle, last) = some_list
print(middle)
(*head, penultimate, last) = some_list
print(head)
```

- Use '_' as placeholder or throwaway variable

List Comprehension

- Provide a concise and easy way to create/transform/filter lists
- Consists of brackets containing an expression followed by for and/or if clauses
- For and if clauses can change places
- Always returns result as a new list
- Basic Syntax
  - [expression **for** item **in** list **if** conditional]
  - [expression **if** conditional **for** item **in** list]

List Comprehension

Double the value of each element in a list

Non Idiomatic:

```
arr = [1, 2, 3, 4, 5, 6]
for item in arr:
    item *= 2
```

Idiomatic:

```
arr = [1, 2, 3, 4, 5, 6]
arr = [x * 2 for x in arr]
```

List Comprehension

Double the value of every even element in a list

Non Idiomatic:

```
arr = [1, 2, 3, 4, 5, 6]
for item in arr:
    if item % 2 == 0:
        item *= 2
```

Idiomatic:

```
arr = [1, 2, 3, 4, 5, 6]
arr = [x * 2 if x % 2 == 0 else x for x in arr]
```

Generator Expressions

- All list comprehensions can be transformed into generator expressions
- Remove the square brackets … DONE

```
arr = [1, 2, 3, 4, 5, 6]
arr = [x * 2 if x % 2 == 0 else x for x in arr]
arr = (x * 2 if x % 2 == 0 else x for x in arr)
```

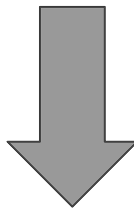
- Why??
- Better and a lot more faster

Updating Lists

```
colors = ['red', 'green', 'blue', 'blue', 'green', 'red', 'red']

del colors[0]
colors.pop(0)
colors.insert(0, 'red')
```

Better Way



```
colors = deque(['red', 'green', 'blue', 'blue', 'green', 'red', 'red'])

del colors[0]
colors.popleft()
colors.appendleft('red')
```

Dictionaries

Use **dict.get()** with **default** parameter to provide default values

Non Idiomatic:

```
debug_level = None
if 'critical' in levels:
    debug_level = levels['critical']
else:
    debug_level = 'info'
```

Idiomatic:

```
debug_level = levels.get('critial', 'info')
```

Counting with Dictionaries

Non-Idiomatic:

```
colors = ['red', 'green', 'blue', 'blue', 'green', 'red', 'red']

d = {}
for color in colors:
    if color not in d:
        d[color] = 0
    d[color] += 1

{'blue': 2, 'green': 2, 'red': 3}
```

Idiomatic:

```
d = {}
for color in colors:
    d[color] = d.get(color, 0) + 1
```

```
from collections import defaultdict
d = defaultdict(int)
for color in colors:
    d[color] += 1
```

Dictionaries

Use **dict comprehensions** to build a dict more efficiently and beautifully

Non Idiomatic:

```
user_address = {}
for user in users:
    if user.address:
        user_address[user.name] = user.address
```

Idiomatic:

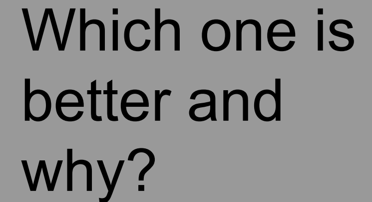
```
user_address = {user.name: user.address for user in users if user.address}
```

Looping Over Dictionary Keys

```
d = {'blue': 2, 'green': 2, 'red': 3}

for k in d:
    print(k)
```

```
for k in d.keys():
    print(k)
    del d[k]
```



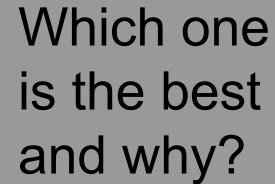
Looping Over Dictionary Keys and Values

```
d = {'blue': 2, 'green': 2, 'red': 3}

for k in d:
    print(k, '...', d[k])
```

```
for k, v in d.items():
    print(k, '...', v)
```

```
for k, v in d.iteritems():
    print(k, '...', v)
```



Cautions!!!

- Idioms and language features are to increase readability and maintainability
- Not to impress other developers
- Do not use them for the sake of using them
- "With power comes great responsibilities"

References and Further Readings

- https://www.pythonforbeginners.com/basics/list-comprehensions-in-python
- https://www.codementor.io/blog/pythonic-code-6yxqdoktzt
- https://www.python.org/dev/peps/pep-0008/
- https://towardsdatascience.com/how-to-be-pythonic-and-why-you-should-care-188d63a5037e
- https://realpython.com/python-code-quality/
- "Writing Idiomatic Python" by Jeff knupp
- Raymond Hettinger Talk: https://www.youtube.com/watch?v=anrOzOapJ2E
- https://www.youtube.com/watch?v=_O23jIXsshs&t=1390s

Happy Idiomatic Coding!!!