Python Idioms

Safe Hammad
Python Northwest
16th January 2014

What is an idiom?

“The specific grammatical, syntactic, and structural character of a given language.”

“A commonly used and understood way of expressing an fact, idea or intention.”

Why care about Python idioms?

"Programs must be written for people to read, and only incidentally for machines to execute."
- Abelson & Sussman, SICP

“There should be one - and preferably only one - obvious way to do it.”
- *Tim Peters, The Zen of Python (PEP 20)*

- The use of commonly understood syntax or coding constructs can aid readability and clarity.
- Some idioms can be faster or use less memory than their “non-idiomatic” counterparts.
- Python's idioms can make your code Pythonic!

Ten idioms

(In no particular order)

1. Make a script both importable and executable

```
if __name__ == '__main__':
```

Example

```
def main():
    print('Doing stuff in module', __name__)

if __name__ == '__main__':
    print('Executed from the command line')
    main()
```

```
$ python mymodule.py
Executed from the command line
Doing stuff in module __main__

>>> import mymodule
>>> mymodule.main()
Doing stuff in module mymodule
```

2. Test for “truthy” and “falsy” values

```
if x:

if not x:
```

Example

```
# GOOD
name = 'Safe'
pets = ['Dog', 'Cat', 'Hamster']
owners = {'Safe': 'Cat', 'George': 'Dog'}
if name and pets and owners:
    print('We have pets!')


# NOT SO GOOD
if name != '' and len(pets) > 0 and owners != {}:
    print('We have pets!')
```

- Checking for truth doesn't tie the conditional expression to the type of object being checked.
- Checking for truth clearly shows the code's intention rather than drawing attention to a specific outcome.

What is truth?

True	False
Non-empty string	Empty string
Number not 0	Number 0
Non-empty container: len(x) > 0	Empty container: len(x) == 0
-	None
True	False
__nonzero__ (2.x) / __bool__ (3.x)	__nonzero__ (2.x) / __bool__ (3.x)

3. Use **in** where possible

Contains:

```
    if x in items:
```

Iteration:

```
    for x in items:
```

Example (contains)

```
# GOOD
name = 'Safe Hammad'
if 'H' in name:
    print('This name has an H in it!')

# NOT SO GOOD
name = 'Safe Hammad'
if name.find('H') != -1:
    print('This name has an H in it!')
```

- Using **in** to check if an item is in a sequence is clear and concise.
- Can be used on lists, dicts (keys), sets, strings, and your own classes by implementing the __contains__ special method.

Example (iteration)

```
# GOOD
pets = ['Dog', 'Cat', 'Hamster']
for pet in pets:
    print('A', pet, 'can be very cute!')

# NOT SO GOOD
pets = ['Dog', 'Cat', 'Hamster']
i = 0
while i < len(pets):
    print('A', pets[i], 'can be very cute!')
    i += 1
```

- Using **in** to for iteration over a sequence is clear and concise.
- Can be used on lists, dicts (keys), sets, strings, and your own classes by implementing the __iter__ special method.

4. Swap values without temp variable

```
a, b = b, a
```

Example

```
# GOOD
a, b = 5, 6
print(a, b)             # 5, 6

a, b = b, a
print(a, b)             # 6, 5

# NOT SO GOOD
a, b = 5, 6
print(a, b)             # 5, 6

temp = a
a = b
b = temp
print(a, b)             # 6, 5
```

- Avoids polluting namespace with temp variable used only once.

5. Build strings using sequence

```
''.join(some_strings)
```

Example

```
# GOOD
chars = ['S', 'a', 'f', 'e']
name = ''.join(chars)
print(name)             # Safe


# NOT SO GOOD
chars = ['S', 'a', 'f', 'e']
name = ''
for char in chars:
    name += char
print(name)             # Safe
```

- The join method called on a string and passed a list of strings takes linear time based on length of list.
- Repeatedly appending to a string using '+' takes quadratic time!

6. EAFP is preferable to LBYL

“It's **E**asier to **A**sk for **F**orgiveness than **P**ermission.”

“**L**ook **B**efore **Y**ou **L**eap”

```
try:            v.     if ...:
except:
```

Example

```
# GOOD
d = {'x': '5'}
try:
    value = int(d['x'])
except (KeyError, TypeError, ValueError):
    value = None

# NOT SO GOOD
d = {'x': '5'}
if 'x' in d and \
   isinstance(d['x'], str) and \
   d['x'].isdigit():
    value = int(d['x'])
else:
    value = None
```

- Throwing exceptions is not “expensive” in Python unlike e.g. Java.
- Rely on duck typing rather than checking for a specific type.

7. Enumerate

```
for i, item in enumerate(items):
```

Example

```
# GOOD
names = ['Safe', 'George', 'Mildred']
for i, name in enumerate(names):
    print(i, name)   #  0 Safe, 1 George etc.

# NOT SO GOOD
names = ['Safe', 'George', 'Mildred']
count = 0
for name in names:
    print(i, name)   #  0 Safe, 1 George etc.
    count += 1
```

- Available since Python 2.3!
- Use the `start` parameter available since Python 2.6 to start at a number other than 0.

8. Build lists using list comprehensions

```
[i * 3 for i in data if i > 10]
```

Example

```
# GOOD
data = [7, 20, 3, 15, 11]
result = [i * 3 for i in data if i > 10]
print(result)   # [60, 45, 33]

# NOT SO GOOD (MOST OF THE TIME)
data = [7, 20, 3, 15, 11]
result = []
for i in data:
    if i > 10:
        result.append(i * 3)
print(result)   # [60, 45, 33]
```

- Very concise syntax.
- Be careful it doesn't get out of hand (in which case the second form can be clearer).

9. Create dict from keys and values using zip

```
d = dict(zip(keys, values))
```

Example

```
# GOOD
keys = ['Safe', 'Bob', 'Thomas']
values = ['Hammad', 'Builder', 'Engine']
d = dict(zip(keys, values))
print(d)  # {'Bob': 'Builder',
             'Safe': 'Hammad',
             'Thomas': 'Engine'}

# NOT SO GOOD
keys = ['Safe', 'Bob', 'Thomas']
values = ['Hammad', 'Builder', 'Engine']
d = {}
for i, key in enumerate(keys):
    d[keys] = values[i]
print(d)   # {'Bob': 'Builder',
              'Safe': 'Hammad',
              'Thomas': 'Engine'}
```

- There are several ways of constructing dicts!

10. And the rest … !

- `while True:`

        break # This will spark discussion!!!
- Generators and generator expressions.
- Avoid `from module import *`

  Prefer: `import numpy as np; import pandas as pd`
- Use _ for "throwaway" variables e.g.:

  `for k, _ in [('a', 1), ('b', 2), ('c', 3)]`
- `dict.get()` and `dict.setdefault()`
- `collections.defaultdict`
- Sort lists using `l.sort(key=key_func)`

''.join(['T', 'h', 'a', 'n', 'k', 's', '!'])

Safe Hammad
http://safehammad.com
@safehammad