Teeing up Python

Code Golf

Lee Sheng
lsheng@yelp.com
@bogosort
Yelp’s Mission
Connecting people with great local businesses.
About Me

Engineer at yelp in London building distributed systems.

Previous stints:

amazon wibi data Dropbox
WARNING

● Not actually a golf player.

● Not talking about traditional “code golfing”

● Monospace type ahead!
Code Golfing

Code Golfing is minimizing the number of strokes in each block of code.

Concise code requires less cognitive load to understand the intent by avoiding implementation errata.

“brevity is the soul of wit” – @WilliamShakespeare
It will take strokes off your code while increasing clarity.

\(^{1}\): No pythons were harmed in the making of this talk.
What are Strokes?

+1: keywords, variable names, each operator (including square brackets)

+0: whitespace, dots, commas, parentheses, quotes, colons, and closing braces/brackets

Effectively counting units of information present.
Why even?

>>> import this

The Zen of Python, by Tim Peters (abridged)

Beautiful is better than ugly.
Simple is better than complex.
If the implementation is easy to explain, it may be a good idea.
Ever written code like this?

to_mail = "UNKNOWN"
if "address" in my_contact:
    to_mail = my_contact["address"]

# Try using a default
to_mail = my_contact.get("address", "UNKNOWN")
Counting up the strokes

to_mail\(^1\) = \"UNKNOWN\(^3\)\"

if\(^4\) "address\(^5\)" in\(^6\) my_contact\(^7\):
    to_mail\(^8\) = my_contact\(^10\)["address\(^12\)"] # 12 strokes

# Try using a default

to_mail\(^1\) = my_contact\(^3\).get\(^4\)("address\(^5\)", "UNKNOWN\(^6\)") # 6 strokes
to_mail = "UNKNOWN"

if "address" in my_contact:
    to_mail = my_contact["address"]

# strokes -= 6
counts = {}
if item not in counts:
    counts[item] = 0
counts[item] += 1

# If only there were a better way!
counts = {}
if item not in counts:
    counts[item] = 0
counts[item] += 1  # 18 strokes

# Why not defaultdict?
from collections import defaultdict  # 4 extra strokes per file

counts = defaultdict(int)
counts[item] += 1  # 13 strokes (including overhead)
Cleaning Up Resources

```
infile = open('myfile.txt', 'r')
for line in infile:
    print(line)
infile.close()

# Why bother explicitly cleaning up?
```
Context Managers

```python
infile = open('myfile.txt', 'r')
for line in infile:
    print(line)
infile.close()  # 13 strokes

# Let’s do this automagically
with open('myfile.txt', 'r') as infile:
    for line in infile:
        print(line)  # 12 strokes
```
try:
    infile = open('myfile.txt', 'r')
    raise Exception()
finally:
    infile.close()  # 12 strokes

# try-finally already baked in by default!
with open('myfile.txt', 'r') as infile:
    raise Exception()  # 9 strokes
“Simple” implementation

# To implement make any class into a context manager, “simply” implement
# the __enter__ and __exit__ methods:

class Tag():
    """Poorly adds html tags""

def __init__(self, name):
    self.name = name

def __enter__(self):
    print("<%s>") % self.name

def __exit__(self, *args):
    print("</%s>") % self.name

# Too much boilerplate, we can do better!
from contextlib import contextmanager

def tag(name):
    "+""Poorly adds html tags"""
    print("<%s>" % name)
    yield  # Do the actual work here
    print("</%s>" % name)

# With enough space to spare, here's an example:
with tag("h1"):
    print("foo")
Functions aren’t scary

def cook(food):
    return 

cook( Millennials) => 🐇

# Lambdas are just functions:
lambda food: 🍿
Quick Functions Primer

```javascript
map([corn, eggs, egg], cook)
=> [rice, pasta, omelette]

filter([rice, pasta, omelette], isVegetarian)
=> [rice, omelette]

reduce([rice, omelette], eat)
=> 🍚
```

Stolen from a tweet from @steveluscher
Goofus runs with the scissors pointing up.

Gallant walks with the scissors pointing down.
Goofus and Gallant explore functions

Goofus thinks iteratively, focusing on how to compute the result.
Goofus has mastered looping over data to compute results.

Gallant thinks functionally, focusing on what the result is.
Gallant has mastered composing functions to compute results.
Goofus iterates over nums, appending doubles of values:

double_nums = []
for n in nums:
    double_nums.append(n * 2)
# 12 strokes

Gallant uses map to compute doubles:

double_nums = list(map(lambda x: x * 2, nums))
# 10 strokes
Goofus and Gallant explore `reduce`

Goofus iterates over `nums`, adding to the total:
```python
total = 0
for n in nums:
    total += n
# 10 strokes
```

Gallant uses a reducer:
```python
total = reduce(lambda x, y: x + y, nums)
# 10 strokes
```
Goofus iterates over nums, appending only evens:

```python
only_evens = []
for n in nums:
    if n % 2 == 0:
        only_evens.append(n)
```

# 16 strokes

Gallant filters nums for evens:

```python
only_evens = list(filter(lambda x: x % 2 == 0, nums))
```

# 12 strokes
BUT WAIT

THERE'S MORE
Comprehending Comprehensions

Comprehensions are a more natural way to construct lists (and dicts).

result = []
for item in things:
    if condition(item):
        result.append(transform(item))
# 14 strokes

result = [ transform(item) for item in things if condition(item) ]
# 12 strokes
Comprehensions Deconstructed

```python
result = []
for item in things:
    if condition(item):
        result.append(transform(item))
```

# strokes -= 2
Better Mapping with Comprehensions

Gallant uses map to produce doubles:

double_nums = list(map(lambda x: x * 2, nums))
# 10 strokes

Billy Mays uses a comprehension:

double_nums = [ x * 2 for x in nums ]
# 10 strokes
Better Filtering with Comprehensions

Gallant filters nums for evens:
only_evens = list(filter(lambda x: x % 2 == 0, nums))
# 12 strokes

Billy Mays uses a comprehension:
only_evens = [ x for x in nums if x % 2 == 0 ]
# 14 strokes
Better Reduces with Comprehensions

Gallant uses a reducer:

```python
total = reduce(lambda x, y: x + y, l)
# 10 strokes
```

Shamwow guy uses the sum function:

```python
total = sum(nums)
# 4 strokes
```
Better dicts with Comprehensions

Goofus iterates, as that’s what he knows:

```
num_to_letters = {}
for x in range(0, 26):
    num_to_letters[x] = chr(97 + x)
# 17 strokes
```

Billy Mays uses a comprehension:

```
num_to_letters = {x: chr(97 + x) for x in range(0, 26)}
# 14 strokes
```
Where can conciseness help?

slides
screens
whiteboards
Quick Whiteboarding Tip

Instead start coding from the upper right, and you can fit 46x11 characters.

If you start coding here, you’ll be awkwardly coding on a 26x6 screen.
Final Takeaways

- Stroke reduction (making code more concise) reduces the cognitive load to understand code.
- Python enables doing more with less.
- For common operations, there’s probably already a builtin or library.
"Je n'ai fait celle-ci plus longue que parce que je n'ai pas eu le loisir de la faire plus courte."

"I apologize for the length of this presentation, but I didn't have time to make it shorter."

- @BlaisePascal
We're Hiring!

www.yelp.com/careers/
talk.exit("That’s all folks!")