Programming Style

CMPT 145

Programming: Style counts

Programs must be written for people to read, and only incidentally for machines to execute.
— Harold Abelson

- For beginners, style is less important than concepts.
- Good style is important for bigger programs.
- A well-written program is easier to debug.
- Most software developers build on software that was written by someone else.
- After six weeks, your own programs will feel like someone else wrote it.

Developing good style

- Don't just write code, read code.
- Learn more about Python.
- Browse `Python.org` and Stack Overflow.
- Learn the basics of any other language.

Improving your code

Turn long if-statements into lookups

- Bad:

```
if month == 1:
    print('Jan')
elif month == 2:
    print('Feb')
# etc, 12 months
```

- Good:

```
month_names = ['Jan', 'Feb', 'Mar', # and the rest
print(month_names[month-1])
```

Improving your code

Turn long if-statements into lookups

- Bad:

```
if month_name == 'Jan':
    month = 1
elif month_name == 'Feb':
    month = 2
# etc, 12 months
```

- Good:

```
month_dict = {'Jan':1, 'Feb':2, 'Mar':3, # and the rest
month = month_dict[month_name]
```

Inproving your code

Factor common statements out.

- Poor:

```
if condition:
  statement A
  statement B
else:
  statement A
  statement C
```

- Better:

```
statement A
if condition:
  statement B
else:
  statement C
```

Inproving your code

Factor common statements out.

- Poor:

```
if condition:
  statement A
  statement B
else:
  statement C
  statement B
```

- Better:

```
if condition:
  statement A
else:
  statement C
statement B
```

Inproving your code

Factor common statements out.

- Poor:

```
for v in range(some):
  statement A # doesn't depend on v
  statement B # depends on v
```

- Better:

```
statement A
for v in range(some):
  statement B # depends on v
```

Improving your code

Don't check for end-of-loop inside your loops!

- Poor:

```
i = 0
while i < N:
     statement B
     if i == N - 1:
         statement A
     i += 1
```

- Better:

```
i = 0
while i < N:
    statement B
     i += 1
statement A
```

Improving your code

Don't use if-statements when a Boolean expression will do.

- Poor:

```python
def function(a,p):
  if a == 3 or p < 0.005:
    return True
  else:
    return False
```

- Better:

```python
def function(a,p):
  return a == 3 or p < 0.005
```

Improving your code

Don't use a while loop for a list, even if you need indices!

- Poor:

```python
i = 0
while i < len(some_list):
    statements about some_list[i]
    i += 1
```

- Better:

```python
for ind, val  in enumerate (some_list):
    statements about some_list[ind]
    statements about val
```

- `enumerate(alist)` returns a sequence of tuples (i, v), where i is the index of the value v in the given list `alist`.

Style Guidelines

Write clearly – don't be too clever

- Poor:

```
new_cl = [[l for l in c if (-1)*flip*first_lit != l]
            for c in clauses if flip*first_lit not in c]
```

- Good:

```
new_clauses = []
for clause in clauses:
    if flip * first_lit not in clause:
        new_clause = []
        for literal in clause:
            if (-1) * flip * first_lit != literal:
                new_clause.append(literal)
        new_clauses.append(clause)
```

Style Guidelines

Break complex expressions into smaller pieces.

- Poor:

```
new_cl = [[l for l in c if (-1)*flip*first_lit != l]
          for c in clauses if flip*first_lit not in c]
```

- Good:

```
new_clauses = []
for clause in clauses:
    if flip * first_lit not in clause:
        new_clause = []
        for literal in clause:
            if (-1) * flip * first_lit != literal:
                new_clause.append(literal)
        new_clauses.append(clause)
```

Style Guidelines

Replace repetitive expressions by calls to a common function

- Poor:

```python
a = 10
b = 12
fa = 1
for i in range(1,a):
    fa = fa * i
fb = 1
for i in range(1,b):
    fb = fb * i
fab = 1
for i in range(1,a+b-1):
    fab = fa * i
print('Mario paths:', fab/(fa*fb))
```

- Good:

```python
def fact(n): # ... assume standard definition
a = 10
b = 12
print('Mario paths:', fact(a+b-2)/(fact(a-1)*fact(b-1)))
```

Style Guidelines

Choose good variable names

- Good names remind the reader of the purpose
- Bad names require the reader to deduce the purpose
- Good:

```
clauses = # ...
left_subtree = # ...
number_of_elements = # ...
```

- Bad:

```
cs = # ...
l = # ...
n = # ...
```

Style Guidelines

Use short names for loop control variables

- Good:

```python
for v in alist:
    # do stuff with v
```

- Bad:

```python
for value_found_in_alist in alist:
    # do stuff with value_found_in_alist
```

Style Guidelines

Avoid using variable names l (ELL) and o (OH)

- Too similar to 1 and 0
- Bad:

```
for l in clauses:
    # do stuff with 1
```

- Better:

```
for cl in clauses:
    # do stuff with cl
```

Style Guidelines

Don't over-comment

- Comments should be about what's not obvious.
- Bad:

```
i = 0 # list index for summing the list
total = 0 # running total for summing the list
while i < len(some_list):
    total = total + i  # add i to the running total
    i += 1   # increment i
```

- Better:

```
# calculate the total cost from the list of costs
i = 0
total = 0
while i < len(some_list):
    total = total + i
    i += 1
```

Advice without examples

Don't patch bad code – rewrite it

- Sometimes there's no way to nudge a program to make it better
- Treat your first implementation as a prototype
- Prototypes get replaced completely all the time
- Learn when you personally need to delete and start again from scratch.

Advice without examples

Making your program faster.

- Make it right before you make it faster
- Keep it right when you make it faster
- Make it clear before you make it faster
- Don't make trivial "optimizations" to make code trivially faster. Think about your algorithm instead!
- Make sure your function is important enough to make faster.

How to be more a more efficient programmer

- Learn to touch type.
- Read as much code as you write.
- Always practice coding. Write little programs every day.
- Keep a journal of the errors that caused you grief.
- Learn more about your language. Not just what gets covered in class.
- Browse Python.org and Stack Overflow. Read. Learn.
- Learn to use the IDE properly.
- Stop using the mouse and menu. Use and memorize key-commands.