

Style

Week 7, day 3

“Coding style”

- Layout
- Largely non-semantic but inferred semantics
- “Code is read more often than it is written”
- Source code is a document in itself
- No one answer

```
def h2(n):  
    if(n==0):  
        return 0  
    if(n ==1) :  
        return 0  
    nv=h2( n-1 )+h2( n-2 )  
    return nv
```

```
def fibonacci (n):  
    """  
    Generate the value of the fibonacci  
    sequence at the given position.  
  
    Each fibonacci number is the sum of the  
    two previous terms and starts 1, 1,  
    2, 3, 5, 8 ...  
    """  
    if (n in (0, 1)):    # start of sequence  
        return 1  
    else:  
        return (fibonacci(n-1) + fibonacci(n-2))
```

Python style

- Already stylish
- Docstrings
- Established (but loose) conventions (see Java)

Names

- “A name is the result of thinking deeply about the ecology of an object”
- Several styles:
 - StudleyCaps or CamelCaps
 - mixedCase
 - under_scores
 - lowercase, UPPERCASE
- Readability

Readability

1. `traverseGraphPostorder`
2. `TraverseGraphPostorder`
3. `traverse_graph_postorder`
4. `TRAVERSEGRAPHPOSTORDER`
5. `TRAVERSE_GRAPH_POSTORDER`
6. `traversegraphpostorder`

Suggestions

- modules: lowercase
- classes: StudleyCaps
- methods: mixedCase or lowercase_underscored
- arguments: lowercase_underscored

Suggestions: variables

- local variables: “the...”
- global variables: “g..”
- constants: UPPERCASE, k
- counter variables: i, j, k item

```
def caicCharToIndex (caic_char):
    """
    Convert a single letter CAIC code into the corresponding clade index.
    """
    ## Preconditions:
    assert (len (caic_char) == 1), "non-char or multiple-char '%s'" % \
        caic_char
    assert (caic_char in _VALID_CAIC_CHARS), "invalid CAIC char '%s'" % \
        caic_char

    ## Main:
    if (caic_char.isupper())
        theIndx = string.ascii_uppercase.find (caic_char)
    elif (caic_char.islower())
        theIndx = string.ascii_lowercase.find (caic_char) + 26
    else:
        assert (caic_char == '-'), "invalid CAIC char '%s'" % caic_char
        theIndx = 52;

    # Postconditions & return:
    assert (0 <= theIndx), "invalid clade index char '%s'" % theIndx
    assert (theIndx <= 52), "invalid clade index '%s'" % theIndx
    return theIndx;
```

Size matters

- A module < 1000 lines (?)
- A single function / method < 1 page
- A block of code ~ 5-7 lines
- Fewer than 5 compulsory arguments to a function (?)

Pythonic idiosyncrasies

- Single line problem
 - Break it up
 - Implicit continue
 - \
 - Lead eye to next line
 - Indent following line
 - 79 characters max

Pythonic idiosyncrasies

2

- Too many levels of indentation
- Tabs & spaces
- Docstrings can be used for tests & documentation.
- Multiline lists & dicts

Multiline lists

```
def plotPolar (
    xdata,          # the x coords
    ydata,          # the y coords
    style,          # tuple of colours
): ...
```

Comments & docstrings

- Yes
- Must agree
- You can overcomment
- Shouldn't be obvious
- Good code is self commenting

This is a good docstring

```
def complex(real=0.0, imag=0.0):  
    """  
    Form a complex number.  
  
    Keyword arguments:  
    real -- the real part (default 0.0)  
    imag -- the imaginary part (default 0.0)  
  
    """  
    if imag == 0.0 and real == 0.0: return complex_zero  
    ...
```

Whitespace

- Is free
- Use lines to break up logical blocks
- Use whitespace around operators
- But not inside brackets
- Seperate args from method

```
if (caic_char.isupper())
    theIndx = string.ascii_uppercase.find (caic_char)
elif (caic_char.islower())
    theIndx = string.ascii_lowercase.find (caic_char) + 26
else:
    assert (caic_char == '-'), "invalid CAIC char '%s'" % caic_char
    theIndx = 52;
```

```
if ( caic_char.isupper ( ) )
    theIndx=string.ascii_uppercase.find(caic_char)
elif ( caic_char.islower ( ) )
    theIndx=string.ascii_lowercase.find(caic_char)+26
else:
    assert (caic_char=='-'), "invalid CAIC char '%s'" % caic_char
    theIndx = 52;
```

Resources

- Ottinger's Rules for Variable and Class Naming". At <http://www.chris-lott.org/resources/cstyle/ottinger-naming.html>
- Guido on Python style <http://www.python.org/dev/peps/pep-0008>
- Docstring conventions <http://www.python.org/dev/peps/pep-0257>

Resources 2

- Another Python style guide <<http://www.cs.indiana.edu/classes/a201-hayn/PythonStyle.html>>
- Python style checker <http://www.cs.caltech.edu/courses/cs11/material/python/misc/python_style_guide.html>