THE ZEN OF PYTHON

Readable Code == Reusable Code

CSCI 3155

Jessica Lynch
Max Harris
Noah Dillon

1. BEAUTIFUL IS BETTER THAN UGLY.

Beautiful code is achieved through a combination of explicit, simple, sparse and flat attributes to ensure readability and therefore the potential for reusability!

```
# Filter elements greater than 4
a = [3, 4, 5]
b = [i for i in a if i > 4]
```

VS.

```
# Filter elements greater than 4
a = [3, 4, 5]
b = []
for i in a:
    if i > 4:
        b.append(i)
```

BEAUTIFUL VS. UGLY

2. EXPLICIT IS BETTER THAN IMPLICIT.

```
def make_complex(x, y):
    return {'x': x, 'y': y}
```

VS.

```
def make_complex(*args):
    x, y = args return
    dict(**locals())
```

EXPLICIT VS. IMPLICIT

3. SIMPLE IS BETTER THAN COMPLEX.

Function with a single recursive call:

```
import os.path as op

def generate_file_list( filepath ):
  pathList = []
  if op.isdir( filepath ):
    for p0 in os.listdir( filepath ):
      path1 = op.join( filepath, p0 )
      if op.isdir( path1 ):
        pathList += generate_file_list( path1 )
      else:
        pathList.append( path1 )
  return pathList
```

VS.

Function with nested recursion:

```
import os.path as op

def generate_file_list( filepath ):
  pathList = []
  for root, dirs, files in os.walk( filepath ):
    for filename in files:
      pathList.append( op.join(root, filename) )
    for dir in dirs:
      generate_file_list( dir )
  return pathList
```

SIMPLE VS. COMPLEX

4. COMPLEX IS BETTER THAN COMPLICATED.

```
import os.path as op

def generate_file_list( filepath ):
   pathList = []
   for root, dirs, files in os.walk( filepath ):
      for filename in files:
         pathList.append( op.join(root, filename) )
      for dir in dirs:
         generate_file_list( dir )
   return pathList
```

VS.

```
import os.path as op

def generate_file_list( filepath ):
   pathList = []
   path0    = filepath
   dirList0 = os.listdir( path0 )
   for p0 in dirList0:
      path1 = op.join( path0, p0 )
      if op.isdir( path1 ):
         dirList1 = os.listdir( path1 )
         for p1 in dirList1:
            path2 = op.join( path1, p1 )
            if op.isdir( path2 )
               dirList2 = os.listdir( path2 )
               for p2 in dirList2:
                  path3 = op.join( path2, p2 )
                  if op.isdir( path3 ):
                     dirList3 = os.listdir( path3 )
                     for p3 in dirList3:
                        path4 = op.join( path3, p3 )
                        if op.isdir( path4 ):
                           dirList4 = os.listdir( path4 )
                           for p4 in dirList4:
                              pathList.append( op.join( path4, p4 ) )
                        else:
                          pathList.append( op.join( path3, p3 ) )
                  else:
                     pathList.append( op.join( path2, p2 ) )
              else:
                 pathList.append( op.join( path1, p1 ) )
      else:
         pathList.append( op.join( path0, p0 ) )
   return pathList
```

COMPLEX VS. **COMPLICATED**

5. FLAT IS BETTER THAN NESTED.

```
def identify(animal):
  if animal.is_vertebrate():
    return identify_vertebrate()
  else:
    return identify_invertebrate()
def identify_vertebrate(animal):
  noise = animal.poke()
  if noise == 'moo':
    return 'cow‘
  elif noise == 'woof':
    return 'dog‘
def identify_invertebrate(animal):
  if animal.is_multicellular():
    return 'Bug!'
  else:
    if animal.is_fungus():
      return 'Yeast'
    else:
      return 'Amoeba'
```

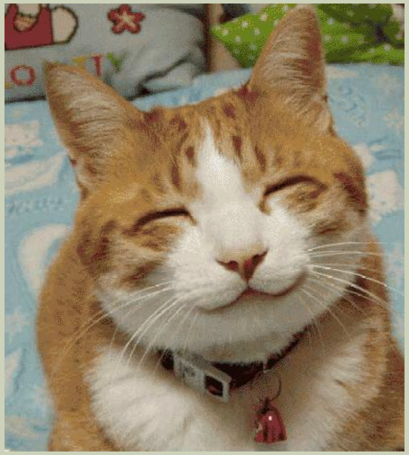
VS.

```
def identify(animal):
  if animal.is_vertebrate():
    noise = animal.poke()
    if noise == 'moo':
      return 'cow‘
    elif noise == 'woof':
      return 'dog‘
    else:
      if animal.is_multicellular():
        return 'Bug!‘
      else:
        if animal.is_fungus():
          return 'Yeast'
        else:
          return 'Amoeba'
```

FLAT VS. NESTED

6. SPARSE IS BETTER THAN DENSE.

One class per module



VS.

Multiple classes per module

SPARSE VS. DENSE

7. READABILITY COUNTS.

"This emphasis on readability is no accident. As an object-oriented language, Python aims to encourage the creation of reusable code… code can hardly be considered reusable if it's not readable. "

– *Guido van Rossum (creator of Python)*

8. SPECIAL CASES AREN'T SPECIAL ENOUGH TO BREAK THE RULES.

Rejected PEP 315: Including do while loops

Conventional way

```
while(True):
    <setup code>
    if(<end condition>):
        break
    <loop code>
```

VS.

Proposed way

```
do:
   <setup code>
while(<end condition>):
   <loop code>
```

GOOD VS. BAD

9. ALTHOUGH, PRACTICALITY BEATS PURITY.

"A Foolish Consistency is the Hobgoblin of Little Minds."

– van Rossum, Warsaw, Coghlan (PEP 8)

10. ERRORS SHOULD NEVER PASS SILENTLY. 11. UNLESS EXPLICITLY SILENCED.

```
try:
    <erroneous code>
except:
    try:
        <fixing code>
    except:
        print <error>
        raise
```

VS.

```
try:
    <erroneous code>
except:
    pass
```

GOOD VS. BAD

12. IN THE FACE OF AMBIGUITY, REFUSE THE TEMPTATION TO GUESS.

```
try:
    with open(fn, 'r') as f:
        lines = list(f)
except (IOError, OSError), err:
    log_error(err)
```

VS.

```
try:
    with open(fn, 'r') as f:
        lines = list(f)
except (IOError, OSError) as err:
    log_error(err)
```

This variable does not leak anymore.

BAD VS. GOOD

13. THERE SHOULD BE ONE -- AND PREFERABLY ONLY ONE -- OBVIOUS WAY TO DO IT.

```
for element in array:
    print element
```

VS.

```
i = 0
while i < len(array):
    print array[i]
    i+=1
```

GOOD VS. JANKY

14. ALTHOUGH, THAT WAY MAY NOT BE OBVIOUS AT FIRST UNLESS YOU'RE DUTCH.

Guido Van Rossum is the creator of Python…
and also, you guessed it, Dutch!

15. NOW IS BETTER THAN NEVER.

16. ALTHOUGH NEVER IS OFTEN BETTER THAN *RIGHT* NOW.

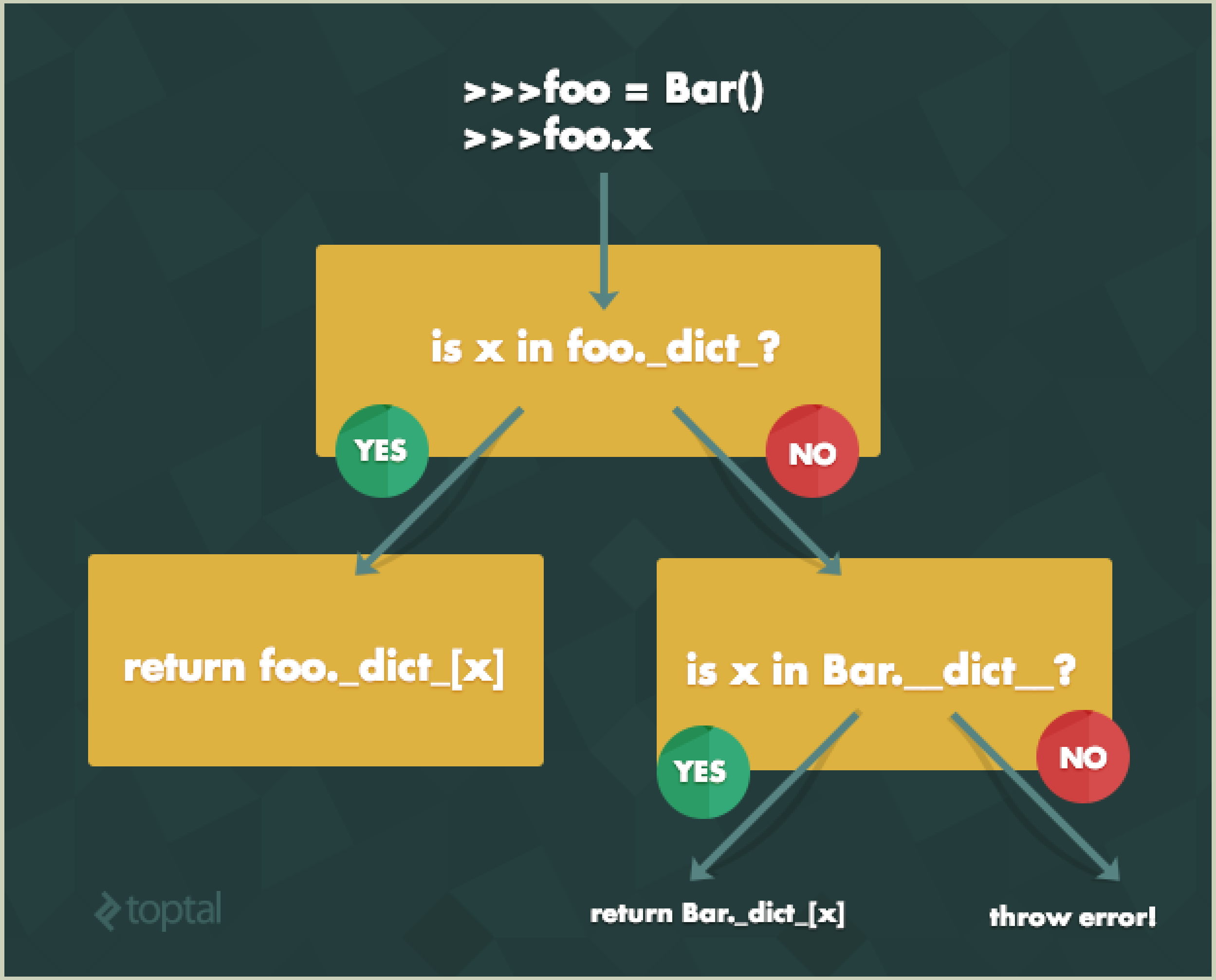
17. IF THE IMPLEMENTATION IS HARD TO EXPLAIN, IT'S A BAD IDEA.

```
def hard():

    import xml.dom.minidom
    document = xml.dom.minidom.parseString(
        '''<menagerie><cat>Fluffers</cat><cat>Cisco</cat></menagerie>''')
    menagerie = document.childNodes[0]
    for node in menagerie.childNodes:
        if node.childNodes[0].nodeValue== 'Cisco' and node.tagName == 'cat':
            return node
```

18. IF THE IMPLEMENTATION IS EASY TO EXPLAIN, IT MAY BE A GOOD IDEA.

```
def easy(maybe):

    import lxml
    menagerie = lxml.etree.fromstring(
        '''<menagerie><cat>Fluffers</cat><cat>Cisco</cat></menagerie>''')
    for pet in menagerie.find('./cat'):
        if pet.text == 'Cisco':
            return pet
```

19. NAMESPACES ARE ONE HONKING GREAT IDEA -- LET'S DO MORE OF THOSE!



REFERENCES

- https://docs.python.org
  - PEPs Index
  - Foreword for "Programming Python" (1st ed.) by Guido Van Rossum
- http://neopythonic.blogspot.com
- http://www.toptal.com/python/python-class-attributes-an-overly-thorough-guide
- http://docs.python-guide.org/en/latest/writing/style
- http://artifex.org/~hblanks/talks/2011/pep20_by_example.html (Blanks, Hunter)