
Python Zen'i
The Zen Of Python
Zen
Attitude

Python'da Esinlenen Felsefeler

- Borrow ideas from elsewhere whenever it makes sense.
- “Things should be as simple as possible, but no simpler.” (Einstein)
- Do one thing well (The "UNIX philosophy").
- Don’t fret too much about performance--plan to optimize later when needed.
- Don’t fight the environment and go with the flow.
- Don’t try for perfection because “good enough” is often just that.
- (Hence) it’s okay to cut corners sometimes, especially if you can do it right later.

Python Zen'i (1) *

- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Sparse is better than dense.
- Readability counts.
- Special cases aren't special enough to break the rules.
- Although practicality beats purity.
- Errors should never pass silently.
- Unless explicitly silenced.

* Tarihçe: http://www.wefearchange.org/2010/06/import-this-and-zen-of-python.html

Python Zen'i (2)

- In the face of ambiguity, refuse the temptation to guess.
- There should be one -- and preferably only one -- obvious way to do it.
- Although that way may not be obvious at first unless you're Dutch.
- Now is better than never.
- Although never is often better than *right* now.
- If the implementation is hard to explain, it's a bad idea.
- If the implementation is easy to explain, it may be a good idea.
- Namespaces are one honking great idea -- let's do more of those!

Beautiful is better than ugly

```
def soundex(string):
    """Returns the Soundex code of the string.
    Characters not A-Z skipped."""

    string=lower(string)

    if not is_letter(string[0]): string=string[1:]
    last=no_tbl[ord(string[0])-97]
    res =upper(string[0])
    # This is where the result will end up

    for char in string[1:]:
if is_letter(char):
   new=no_tbl[ord(char)-97]
   if (new!="0" and new!=last):
res=res+new
   last=new



    if len(res)<4:
return res+"0"*(4-len(res))
    else:
return res[:4]
```

```
sub soundex
{
  local (@s, $f, $fc, $_) = @_;

  push @s, '' unless @s;
  # handle no args as a single empty string

  foreach (@s)
  {
    tr/a-z/A-Z/;
    tr/A-Z//cd;

    if ($_ eq '')
    {
      $_ = $soundex_nocode;
    }
    else
    {
      ($f) = /^(.)/;
      tr/AEHIOUWYBFPVCGJKQSXZDTLMNR/00000000111122222222334556/;
      ($fc) = /^(.)/;
      s/^$fc+//;
      tr///cs;
      tr/0//d;
      $_ = $f . $_ . '000';
      s/^(.{4}).*/$1/;
    }
  }

  wantarray ? @s : shift @s;
}
```

Explicit is better than implicit

Yanlış:

```
from os import *
print getcwd()
```

Doğru:

```
import os
print os.getcwd()
```

Explicit Self:

```
class Bag:
    def __init__(self):
        self.data = []
    def add(self, x):
        self.data.append(x)
    def addtwice(self, x):
        self.add(x)
        self.add(x)
```

Simple is better than complex

Java / Php Kodu:

```
if ((my_array == null) || (my_array.length == 0)) {
  # process my array
}
```

Python Kodu:

```
if not my_array:
  # process my array
```

Complex is better than complicated

Complicated:
Difficult to analyze or understand

Complex:
Composed of many interconnected parts; compound; composite

```
women = complicated

software = complex
```

Flat is better than nested

Nested:

```
if foo:
  return bar
else:
  baz = fie(fum)
  return baz + blab
```

Flat:

```
if foo:
  return bar
baz = fie(fum)
return baz + blab
```

Nested:

```
for item in container:
  if interesting(item):
    dothis(item)
    dothat(item)
    theother(item)
```

Flat:

```
for item in container:
  if not interesting(item):
    continue
  dothis(item)
  dothat(item)
  theother(item)
```

Sparse is better than dense

Doğru sözdizimi:

```
if i>0: return sqrt(i)
elif i==0: return 0
else: return 1j * sqrt(-i)
```

Ama neden kasalım?

```
if i > 0:
    return sqrt(i)
elif i == 0:
    return 0
else:
    return 1j * sqrt(-i)
```

Readability Counts

C Kodu 1:

```
#include <stdio.h>
int main(void) {
     printf("Merhaba,dünya!\n");
     return(0);
}
```

C Kodu 2:

```
#include <stdio.h> int main(void)
{ printf("Merhaba,
dünya!\n"); return(0); }
```

Python Kodu:

```
print “Merhaba,dünya!”
```

Geleceğe Bakış

```
>>> from __future__ import braces
File "<stdin>", line 1
SyntaxError: not a chance
```

Special cases aren't special enough to break the rules

```
>>>0.1
0.10000000000000001

>>>1/2
0

>>>1.0/2
0.5
```

@staticmethod, @classmethod da basit bir wrapper olduğu için farklı bir syntax geliştirmeye gerek yok.

Although Practicality Beats Purity

```
object.len yerine len(object)
integer.to_str yerine str(integer)
string.to_int yerine int(string)
float()
chr()
ord()
...
...
```

Errors should never pass silently

Örnek 1:

```
>>>import mymodule

ImportError                               Traceback (most recent call last)
/home/amiroff/<ipython console> in <module>()
ImportError: No module named mymodule
```

Örnek 2:

```
>>>len(3)
TypeError                                 Traceback (most recent call last)
/home/amiroff/<ipython console> in <module>()
TypeError: object of type 'int' has no len()
```

Unless explicitly silenced

Örnek 1:

```
try:
    import mymodule
except ImportError:
    print 'this module is not available'
    # pass
```

Örnek 2:

```
try:
    v = d[k]
except KeyError:
    v = d[k] = default
```

In the face of ambiguity, refuse the temptation to guess

Bu kod ne yapar?

```
if not a and b:
    pass
```

Bunu mu? (Hayır)

```
if not (a and b):
    pass
```

Kod bizi tahmine zorlamamalıdır:

```
if b and not a:
    pass
```

Veya:

```
if (not a) and b:
    pass
```

There should be one -- and preferably only one -- obvious way to do it.

Python Kodu:

```
>>> [(x, x*x) for x in [1,2,3,4] if x != 3]
[(1,1), (2,4), (4,16)]
```

Ruby Kodu 1:

```
[1,2,3,4].select {|x| x != 3} . map! {|x| [x, x*x]}
```

Ruby Kodu 2:

```
[1,2,3,4].map! { |x| [x,x*x] if x != 3 }.compact!
```

Ruby Kodu 3:

```
[1,2,3,4].inject([]) { |a,x| ( a << [x, x*x] if x != 3 ) || a }
```

Ruby Kodu 4:

```
[1,2,3,4].inject([]) {|a,x| x == 3 ? a : a << [x, x*x]}
```

Although that way may not be obvious at first unless you're Dutch.

C Kodu:

```
a = cond ? expr1 : expr2
```

Python Kodu:

```
a = expr1 if cond else expr2
```

Now is better than never

- Python 2.x ve 3.x
- from __future__ import FeatureName

Although never is often better than *right* now

Hazır olmadan bir özelliği geliştirmek hiç geliştirmemekten iyi olsa da, herşeyi bırakarak tek özelliğin geliştirilmesine odaklanmak da bazen bu özelliğin geliştirilmemesinden daha kötü olabilir.

If the implementation is hard to explain, it's a bad idea.

If the implementation is easy to explain, it may be a good idea.

Namespaces are one honking great idea -- let's do more of those!

C# Kodu:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;
using System.Web.Mvc;
```

Ruby Kodu:

```
require 'active_support'
require 'active_support/i18n'
require 'active_model'
require 'arel'

module ActiveRecord
  extend ActiveSupport::Autoload
  ....
end
```

Python Kodu:

```
# Doğru:
import module
breakfast = module.SpamAndEggs()

# Yanlış:
from module import *
```

Foto: http://www.flickr.com/photos/audiotribe/3332139778/