Developer Days
Network Services Orchestrator

Zen of Python

Harish Ravichandran, Software Consulting Engineer
Raghul Prasanth, Consulting Engineer

Introduction

- Poem by Tim Peters
- Cast the spell “import this”
- Mini style-guide for Python coders

Beautiful is better than ugly

```
final_items = [process(item) for subitems in items
                             for subitems2 in subitems
                             for item in subitems2
                             if item == "correct"]
```

```
def select_correct_items(items):
    correct = []
    for subitems in items:
        for subitems2 in subitems:
            correct = [item for item in subitems2 if item == "correct"]
    return correct

final_items = select_correct_items(items)
```

Explicit is better than implicit

```
from math import sin
def sin(a):
    return "Not Commenting is a sin"

print(sin(1))
```

```
import math
def sin(an_integer):
    return "Not commenting is a sin"

print(sin(1))

print(math.sin(1))
```

Simple is better than complex
Complex is better than complicated

```python
def recursion_reverse(s):
    # Base Case
    if(len(s)==1):
        return s
    else:
        return recursion_reverse(s[1:]) + s[0]

original_string = "IloveCisco"
print("Original String : ", original_string)
print("Reversed String : ", recursion_reverse(original_string))
```

```python
my_string = "IloveCisco"
reversed_string = my_string[::-1]
```

Flat is better than nested

```
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'
```

```
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'
```

Sparse is better than dense

```
print('\n'.join("%i  bytes = %i bits which has %i possible values."
% (j, j*8, 256**j-1) for  j in (1 << i for i in range(8))))
```

```
for i in range(8):
    j = 1<<i
    byte = j
    bits = j*8
    values = 256**j-1
    print ("{} bytes = {} bits which has {} possible values"
            .format(byte, bits, values))
```

Readability counts

```
import time
import os

bfile = '/path/to/birthday/file'

def check():
    fileName = open(bfile, 'r')
    today = time.strftime('%m%d')
    f = 0
    for line in fileName:
        if today in line:
            line = line.split(' ')
            f =1
            os.system('notify-send "Birthdays Today: ' + line[1]
            + ' ' + line[2] + '"')
    if f == 0:
            os.system('notify-send "No Birthdays Today!"')

if __name__ == '__main__':
    check()
```

```
# Python program For Birthday Reminder Application
import time
import os

# Birthday file is the one in which the actual birthdays
# and dates are present. This file can be
# manually edited or can be automated.
# For simplicity, we will edit it manually.
# Birthdays should be written in this file in
# the format: "MonthDay Name Surname" (Without Quotes)

birthdayFile = '/path/to/birthday/file'

def checkTodaysBirthdays():
    fileName = open(birthdayFile, 'r')
    today = time.strftime('%m%d')
    flag = 0
    for line in fileName:
        if today in line:
            line = line.split(' ')
            flag =1
            # line[1] contains Name and line[2] contains Surname
            os.system('notify-send "Birthdays Today: ' + line[1]
            + ' ' + line[2] + '"')
    if flag == 0:
            os.system('notify-send "No Birthdays Today!"')

if __name__ == '__main__':
    checkTodaysBirthdays()
```

Special cases aren't special enough to break the rules. Although practicality beats purity

```
class cal():
    def __init__(self,a,b):
        self.a=a
        self.b=b

    @property
    def a(self):
        return self._a

    @property
    def b(self):
        return self._b

     @a.setter
     def a(self, a):
         self._a = a

     @b.setter
     def b(self, b):
         self._b = b

    def add(self):
        return self.a+self.b

a=int(input("Enter first number: "))
b=int(input("Enter second number: "))
obj=cal(a,b)
print("Result: ",obj.add())
```

```
def add(a, b):
    return a + b

a=int(input("Enter first number: "))
b=int(input("Enter second number: "))
print("Result: ",add(a, b))
```

Errors should never pass silently

```
def divide(a, b):
    if b == 0:
        result = None
    else:
        result = a/b

    return result
```

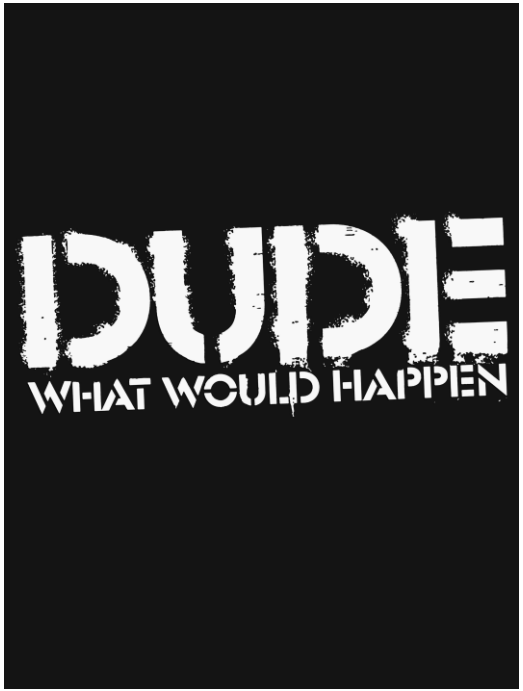
```
def divide(a, b):
    if b == 0:
        raise Exception("Divide by zero not permitted")
    result = a/b

    return result
```

Unless explicitly silenced

```
if key not in value:
    raise Exception("Key error")
else:
    value[key] = value[key] + 10
```

```
try:
    value[key] = value[key] + 10
except KeyError:
    value[key] = 10
```

In the face of ambiguity, refuse the temptation to guess.



What would be printed on the screen?

```
a = True
b = True
if not a and b:
    print('Hello World')
else:
    print('Bye World')
```

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

This belief is throwing shade on the Perl programming language's motto,

"There's more than one way to do it!"

Whether to use postfix or prefix operator??????

Python solves this confusion by not supporting them altogether.

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

This aphorism refers to the creator of Python, Guido van Rossum, who is Dutch. A common example used to denote his genius is C's hotly debated ternary operator.

```
a = condition?expression1:expression2;
```

In Python,

```
a = expression1 if condition else expression2
```

However, not even this aphorism prevented Python from incorporating three different ways of formatting strings.

Now is better than never. Although never is often better than *right* now.

It's almost certainly **better to wait for your program to finish**, than to have it finish too early with incorrect results.

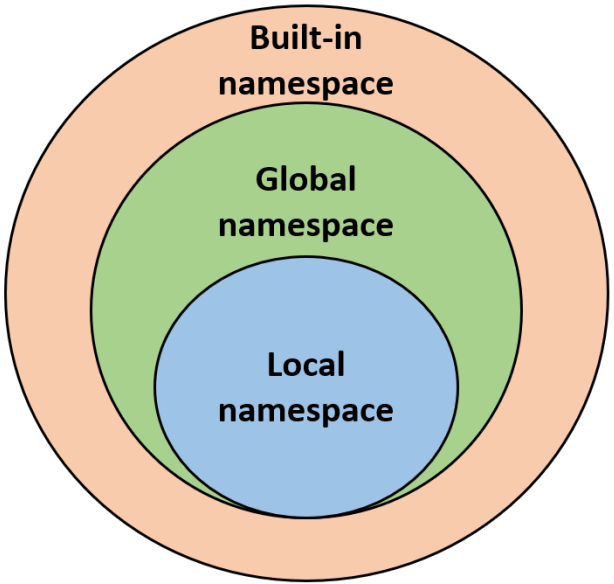
It's best to start today, rather than procrastinating endlessly. But a project never undertaken might **still be better than a hastily executed one**.

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.

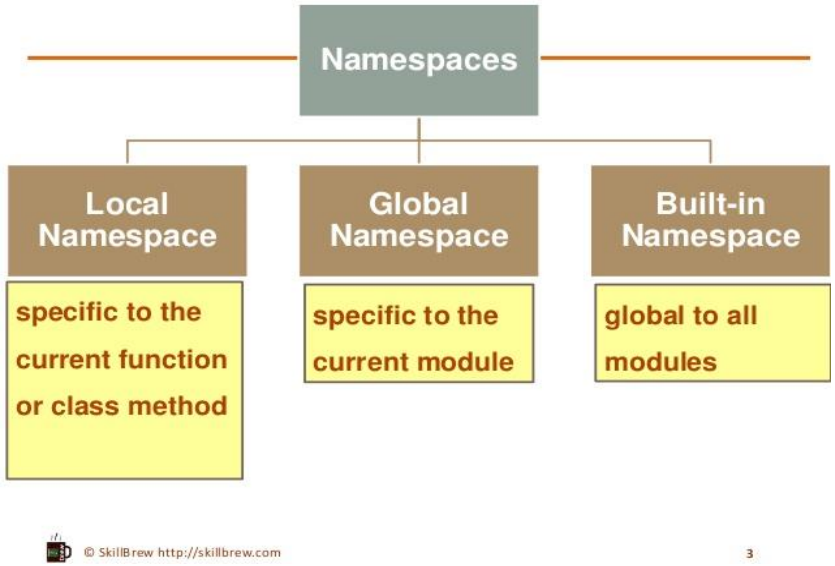
Python strives to make the programmer's job easier rather than accommodate the computer so a program runs faster.

"High-Performance" code is so complicated as to be impossible for programmers to understand and debug, then it's bad code.

Namespaces are one honking great idea...



Type of Namespaces



Conclusion

The underlying philosophy remains:
“practicality beats purity”