Стили в Python: как перестать рвать на себе волосы, читая чужой код (Стилёчек)

Мосягин Игорь

Кто я

Пишу на питоне с ~2009 года

Работал бэкендером, в науке, дата сайнтистом, дата инженером, ML инженером

Писал код в продакшн ещё на C/C++/Fortran/Golang

Хакатоню, опенсоршу

Преподаю

Вижу страдающих студентов, друзей и коллег которые грызут литкод

Хочу предложить ещё идей как развиваться

Вопросы опыта и (само)рефлексии

Если сеньору надо добавить функциональность в код, стиль которого не нравится линтеру, что сделает сеньор?

А что сделает миддл в такой ситуации?

Какая частая проблема Python-разработчиков здесь рифмуется?

О чём будет доклад

В целом, о стилях как способе сделать себя более гибким

1: О том, какие стили бывают и у всех на слуху

2: О том, какие стили бывают ещё

3: О том, как вы можете найти свой стиль

Закончим рассуждениями о том, что с этим делать

что за стили

Стили кода

У нас есть PEP-0008

У нас есть psf/black

Мы часто считаем что только наш выбор верный

👉 статья от Ильи Лебедева про названия (в конце QR на ссылки)

Какие стили программирования вы знаете?

Общеизвестны три

* Процедурный
* ООП
* Функциональный

Аспекты и модификаторы

Читаемость

Расширяемость

Краткость

Производительность

Применимость

Модельная задача

Вордкаунт

```
мама мыла раму мама мама
->
мама 3
мыла 1
раму 1
```

есть ещё файл со стоп-словами

без нормализации слов

всё влезает в память

никаких сложных требований

Процедурная каша

```
def count_words_procedural(text):
  words = text.lower().split()
  word_count = {}

  for word in words:
    if word in word_count:
      word_count[word] += 1
    else:
      word_count[word] = 1

  sorted_words = sorted(word_count.items(), key=lambda x: x[1],
reverse=True)

  for word, count in sorted_words[:5]:
    print(f"{word}: {count}")
```

"ООП головного мозга"

```
class WordCounter:

  def __init__(self, text):
    self.words = text.lower().split()
    self.word_count = {}

  def count_words(self):
    for word in self.words:
      self.word_count[word] = self.word_count.get(word, 0) + 1

  def get_top_words(self, n=5):
    return sorted(
      self.word_count.items(),
      key=lambda x: x[1],
      reverse=True
    )[:n]

  def print_top_words(self, n=5):
    for word, count in self.get_top_words(n):
      print(f"{word}: {count}")
```

Функциональщина

```
from collections import Counter
from functools import partial

def count_words_functional(text):
  words = text.lower().split()
  return Counter(words)

def get_top_words(word_counts, n=5):
  return word_counts.most_common(n)

def print_word_counts(word_counts):
  for word, count in word_counts:
    print(f"{word}: {count}")

process_text = lambda text:
print_word_counts(get_top_words(count_words_functional(t)))
```

ФП: "Сеньор скучал" aka write-only

```
import re, sys, collections


stopwords = set(open('../stop_words.txt').read().split(','))

words = re.findall('[a-z]{2,}', open(sys.argv[1]).read().lower())

counts = collections.Counter(w for w in words if w not in stopwords)

for (w, c) in counts.most_common(25):
  print(w, '-', c)
```

дальше — хуже

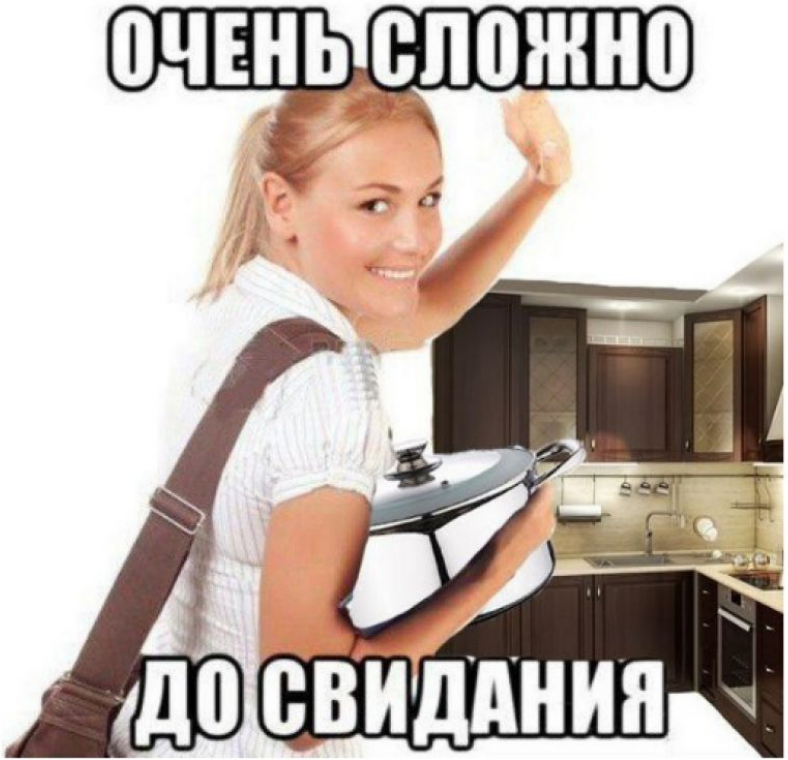
"Я узнал что такое Монада"

```
import sys, re, operator, string

class TFTheOne:
  def __init__(self, v):
    self._value = v
  def bind(self, func):
    self._value = func(self._value)
    return self
  def printme(self):
    print(self._value)

def read_file(path_to_file):
  with open(path_to_file) as f:
    data = f.read()
  return data

def filter_chars(str_data):
  pattern = re.compile('[\W_]+')
  return pattern.sub(' ', str_data)

def normalize(str_data):
  return str_data.lower()

def scan(str_data):
  return str_data.split()

def remove_stop_words(word_list):
  with open('../stop_words.txt') as f:
    stop_words = f.read().strip('\n').split(',')
    stop_words.extend(list(string.ascii_lowercase))
  return [w for w in word_list if not w in stop_words]
```

```
def frequencies(word_list):
  word_freqs = {}
  for w in word_list:
    if w in word_freqs:
      word_freqs[w] += 1
    else:
      word_freqs[w] = 1
  return word_freqs

def sort(word_freq):
  return sorted(word_freq.items(),
key=operator.itemgetter(1), reverse=True)

def top25_freqs(word_freqs):
  top25 = ""
  for tf in word_freqs[0:25]:
    top25 += str(tf[0]) + ' - ' + str(tf[1]) + '\n'
  return top25

TFTheOne(sys.argv[1]).bind(read_file)\
.bind(filter_chars).bind(normalize).bind(scan)\
.bind(remove_stop_words).bind(frequencies).bind(sort)\
.bind(top25_freqs)\
.printme()
```

"Я узнал что такое Монада"

```
import sys, re, operator

class TFTheOne:
  def __init__(self, v):
    self._value = v
  def bind(self, func):
    self._value = func(s
    return self
  def printme(self):
    print(self._value)

def read_file(path_to_fi
  with open(path_to_file
    data = f.read()
  return data

def filter_chars(str_data
  pattern = re.compile('
  return pattern.sub(' ',

def normalize(str_data):
  return str_data.lower(

def scan(str_data):
  return str_data.split(

def remove_stop_words(wor
  with open('../stop_word
    stop_words = f.read(
    stop_words.extend(lis
  return [w for w in word
```




```
(),
erse=True)


 + str(tf[1]) + '\n'


_file)\
ize).bind(scan)\
requencies).bind(sort)\
```

Когда облиткодился

```
import heapq, re, sys

ww = re.findall("[a-z]{2,}", open(sys.argv[1]).read().lower())
for w in heapq.nlargest(25, set(ww) -
set(open("../stop_words.txt").read().split(",")), ww.count):
  print(w, '-', ww.count(w))
```

Я пришёл писать код и пить кофе и кофе я уже выпил

```
import sys, string
word_freqs = []
with open('../stop_words.txt') as f:
  stop_words = f.read().split(',')
  stop_words.extend(list(string.ascii_lowercase))

for line in open(sys.argv[1]):
  start_char = None
  i = 0
  for c in line:
    if start_char == None:
      if c.isalnum():
        start_char = i
      else:
        if not c.isalnum():
          found = False
          word = line[start_char:i].lower()
        if word not in stop_words:
          pair_index = 0
          for pair in word_freqs:
            if word == pair[0]:
              pair[1] += 1
              found = True
              break
            pair_index += 1
          if not found:
            word_freqs.append([word, 1])
          elif len(word_freqs) > 1:
            for n in reversed(range(pair_index)):
              if word_freqs[pair_index][1] > word_freqs[n][1]:
                word_freqs[n], word_freqs[pair_index] = word_freqs[pair_index], word_freqs[n]
                pair_index = n
        start_char = None
    i += 1

for tf in word_freqs[0:25]:
  print(tf[0], '-', tf[1])
```

Вернёмся к аспектам

Читаемость: какой стиль легче понять с первого взгляда?

Расширяемость: как легко добавить новую функциональность в каждом стиле?

Краткость: сравните количество строк кода в каждом стиле

Производительность: есть ли разница в эффективности между стилями?

Применимость: в каких ситуациях каждый стиль может быть предпочтительнее?

и что?

Разберём кусочек кода, представьте что вы ревьювер

Пример 1. Самый простой пример это функции в которых нет никаких ветвлений:

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

Нет ветвлений, дебажить легко

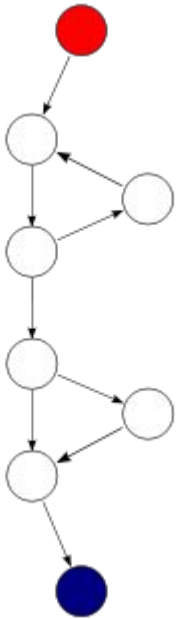
Пример 2. В коде функции есть условие

```
def is_even(number):
      if number % 2 == 0:
            return True
      else:
            return False
```

Пример 3. Когда в функции накрутили

```
def count_even(numbers):
      count = 0
      for number in numbers:
            if number % 2 == 0:
                  count += 1
            elif number < 0:
                  break
      return count
```

Цикломатическая сложность



СЛОЖНОСТЬ = [количество рёбер] - [количество узлов] + 2
Количество рёбер это количество стрелок на картинке
Количество узлов это количество всех кружочков на картинке

Получится какое-то число, обычно в диапазоне от 1 до 20-30 (меньше = лучше)

На картинке 9 стрелочек, 8 кружочков и если есть какой-то код представляемый таким графом выполнения то его цикломатическая сложность 9 - 8 + 2*1 = 3

Границы сложности в 1976 году от Тома МакКейба

- 1-10: простой код, низкий риск проблем
- 11-20: код посложнее, средний риск неприятностей
- 21-50: совсем сложно, высокий риск проблем
- >50: очень сложный код который невозможно проверить *жесть*

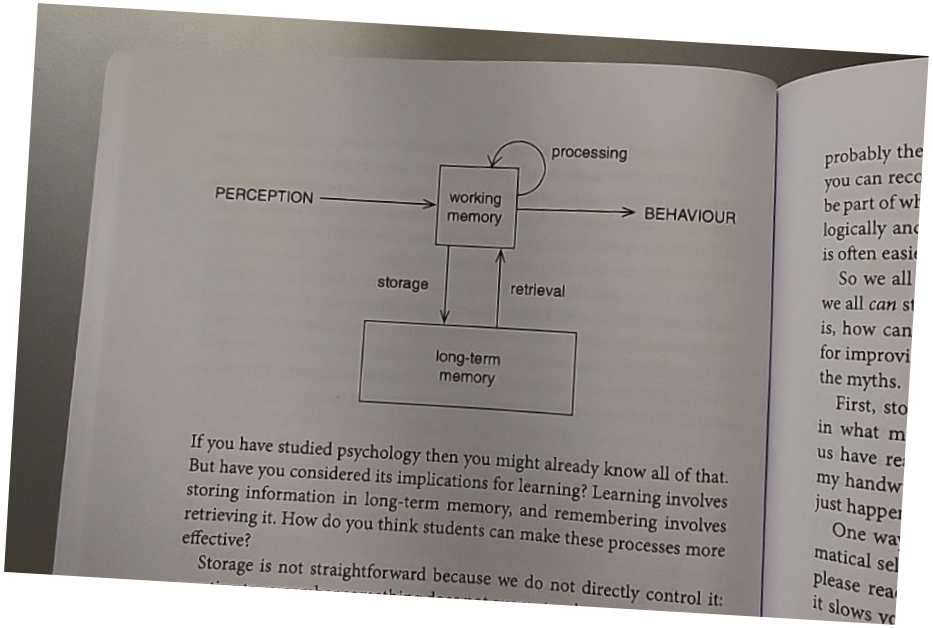
https://github.com/PyCQA/mccabe
https://radon.readthedocs.io/en/latest/intro.html

Какой из этих двух примеров имеет выше сложность?

```
import sys, string
word_freqs = []
with open('../stop_words.txt') as f:
  stop_words = f.read().split(',')
  stop_words.extend(list(string.ascii_lowercase))

for line in open(sys.argv[1]):
  start_char = None
  i = 0
  for c in line:
    if start_char == None:
      if c.isalnum():
        start_char = i
      else:
        if not c.isalnum():
          found = False
          word = line[start_char:i].lower()
        if word not in stop_words:
          pair_index = 0
          for pair in word_freqs:
            if word == pair[0]:
              pair[1] += 1
              found = True
              break
            pair_index += 1
          if not found:
            word_freqs.append([word, 1])
          elif len(word_freqs) > 1:
            for n in reversed(range(pair_index)):
              if word_freqs[pair_index][1] > word_freqs[n][1]:
                word_freqs[n], word_freqs[pair_index] = word_freqs[pair_index], word_freqs[n]
                pair_index = n
        start_char = None
    i += 1

for tf in word_freqs[0:25]:
  print(tf[0], '-', tf[1])
```

```
def count_words_procedural(text):
  words = text.lower().split()
  word_count = {}

  for word in words:
    if word in word_count:
      word_count[word] += 1
    else:
      word_count[word] = 1

  sorted_words = sorted(
      word_count.items(),
      key=lambda x: x[1],
      reverse=True,
  )

  for word, count in sorted_words[:5]:
    print(f"{word}: {count}")
```

Когнитивную нагрузка и обучение



Почему сеньорам проще писать сложно?

- не надо лезть "далеко" за структурами

- есть личная коллекция трюков

- опыт: уже делали что-то такое и знают что получится

Используемые сущности упакованы в ментальные кирпичики

🤔 У менее опытного разработчика этого может не быть

Стили = то что мы хотим видеть в росте инженера

1 Стили программирования как "линзы" для рассмотрения проблем

2 Расширение "ментального инструментария" разработчика

3 Влияние на процесс проектирования решений

4 Стили как способ структурирования кода и мыслей

5 Развитие абстрактного мышления

6 Преодоление ограничений одного стиля мышления

7 Стимуляция креативности в программировании

Какие *внешние* бонусы если знаешь разные стили?

1. Адаптация к существующим проекта
2. Снова можешь "набросать" скриптик
3. Улучшение коммуникации
4. Повышение качества код-ревью
5. Гибкость в решении проблем

Что мне с этим делать?

Стили важнее чем литкод

Стили актуальны независимо от технологии

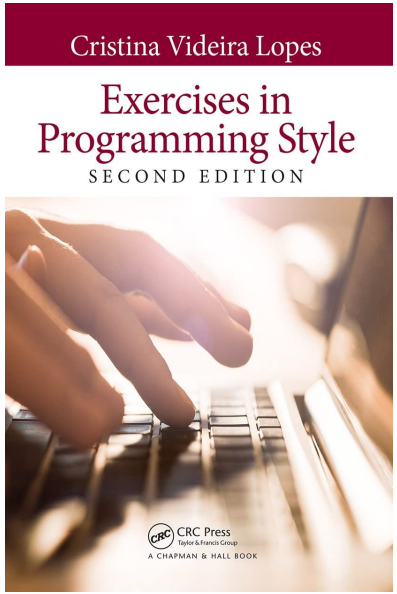
Каждый стиль это новый способ мышления, хотя бы чуток

Мастерство в разнообразии

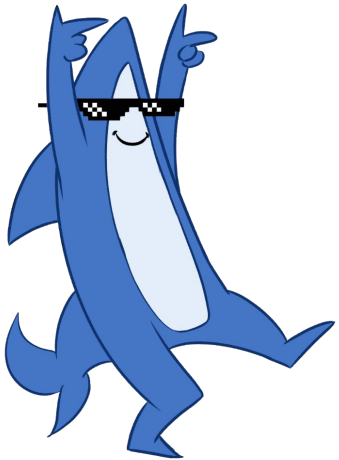
Решайте знакомые задачи новыми способами

Найди свой стиль

пишите, пишите, пишите



@shrimpsizemoose



ссылки и материалы