

ПИТОН

Четверг

# Использование руководства сообщества Python по стилю написания кода.

- Согласно Дзен языка Python, "должен существовать один — и желательно только один — очевидный способ сделать это". В духе предоставления единственного очевидного "правильного способа" делать вещи и в целях достижения консенсуса вокруг этих практических приемов сообщество Python выпускает рекомендации по улучшению языка Python, которые представляют собой правила написания программного кода на Python, в состав которых входит стандартная библиотека главного дистрибутива Python.

# Использование руководства сообщества Python по стилю написания кода.

- Наиболее важными из них являются рекомендации PEP 8, руководство по написанию программного кода на языке Python. Время идет, и PEP 8 регулярно эволюционирует, поскольку выявляются новые правила, а прошлые устаревают из-за изменений в языке.

# Использование руководства сообщества Python по стилю написания кода.

- Рекомендации PEP 8 устанавливают стандарты для правил именования, использования пустых строк, отступов и пробелов, максимальной длины строки, комментариев и т. д. Цель состоит в том, чтобы улучшить читаемость кода и сделать его единообразным между широким спектром программ на Python. Когда вы только начинаете программировать, то должны стремиться научиться и следовать принятым правилам до того, как укоренятся вредные привычки.

# Использование руководства сообщества Python по стилю написания кода.

- Программный код в этой книге будет точно соответствовать рекомендациям PEP 8, но из уважения к издательской индустрии я переопределил некоторые правила (например, за счет меньшего объема комментированного кода, меньшего числа пустых строк и более коротких литералов документирования).

# Использование руководства сообщества Python по стилю написания кода.

- Стандартизированные имена и процедуры особенно важны, когда вы работаете в кросс-функциональных группах. При переводе с языка ученых на язык инженеров многое может потеряться, как в 1999 г., когда инженеры потеряли климатический орбитальный спутник Марса, потому что разные группы разработчиков использовали разные единицы измерения. В течение почти двух десятилетий я строил компьютерные модели Земли, которые трансформировались в инженерную функцию.

# Использование руководства сообщества Python по стилю написания кода.

- Инженеры использовали мои скрипты для загрузки этих моделей в собственные программы. От проекта к проекту они делились этими скриптами между собой, тем самым повышая эффективность и помогая неопытным. Поскольку эти "командные файлы" были специально настроены под каждый проект, то по понятным причинам инженеры были не в восторге, когда во время обновлений моделей имена атрибутов менялись. По сути дела, одним из их внутренних принципов было "Упрашивай, подкупай или запугивай— лишь бы твой разработчик моделей применял единообразные имена свойств!".

# Возвращаемся к функциям

- Реализовать следующую задачу в виде функции: необходимо создать функцию, которая будет принимать параметр строки и проверять есть ли в принятой строке пробел. Если есть, то возвращаем строку в верхнем регистре, а если нет, то в нижнем.

```
19 def get_string(s):
20     if ' ' in s:
21         return s.upper()
22     else:
23         return s.lower()
24
25
26 print(get_string("Hello"))
27 print(get_string("Hello, world!"))
28 |
```



# Возвращаемся к функциям

```
18
19 def get_string(s):
20     if ' ' in s:
21         return s.upper()
22     else:
23         return s.lower()
24
25
26 print(get_string(input()))
27 print(get_string(input()))
28 |
```

Run: randomizer\_names x

```
C:\Users\Endliar\PycharmProjects\guess_the_number_game\venv\Scripts\python.exe C:/Users/Endliar/PycharmP
hello, world
HELLO, WORLD
HELLOWORLD
helloworld
```

# Возвращаемся к функциям

- Реализуем задачу суммы всех переменных аргументов.

```
29
30 def get_sum(x, y, c, d):
31     return x + y + c + d
32
33
34 print(get_sum(4, 2, 4, 2))
35 |
```

# Возвращаемся к функциям

- А если сделаем так?

```
10  
11 def get_sum(a, b, c=0, d=1):  
12     return a + b + c + d  
13  
14  
15 print(get_sum(1, 2))
```

# Переменное количество аргументов

```
29
30
31 def f(a, x, *args, **kwargs):
32     print(a)
33     print(x)
34     print(args)
35     print(kwargs)
36
37
38 f(1, 2, 3, 4, b='test', c='hi')
39
```

# Переменное количество аргументов

```
35
36 def get_sum(*args, **kwargs): # именованные и позиционные аргументы
37     print(args) # если передадим любое n-ое кол-во аргументов
38
39
40     get_sum(1, 5, 10) # то увидим, что аргументы которые нам передаются - это действительно кортеж
41
42
43 def func(**kwargs): # возвращает аргументы словарями
44     print(kwargs)
45
46     func(a = 10, b = 5, c = 20)
```