Документирование кода. Модули.

Документирование кода в python - повышает читаемость и быстроту понимания кода, как другими людьми, так и самим автором.

Соглашения, связанные со строками документации python, описывает PEP 257

Раздаточный материал № 165 (справочно)

Однострочные

Нет пустых строк перед или после документации.

Используйте тройные кавычки, даже если документация умещается на одной строке. Потом будет проще её дополнить.

Закрывающие кавычки на той же строке. Это смотрится лучше.

Нет пустых строк перед или после документации.

Однострочная строка документации не должна быть "подписью" параметров функции / метода (которые могут быть получены с помощью интроспекции).

Вставляйте пустую строку до и после всех строк документации (однострочных или многострочных), которые документируют класс - вообще говоря, методы класса разделены друг от друга одной пустой строкой, а строка документации должна быть смещена от первого метода пустой строкой; для симметрии, поставьте пустую строку между заголовком класса и строкой документации. Строки документации функций и методов, как правило, не имеют этого требования.

Строки документации скрипта (самостоятельной программы) должны быть доступны в качестве "сообщения по использованию", напечатанной, когда программа вызывается с некорректными или отсутствующими аргументами (или, возможно, с опцией "-h", для помощи). Такая строка документации должна документировать функции программы и синтаксис командной строки, переменные окружения и файлы. Сообщение по использованию может быть довольно сложным (несколько экранов) и должно быть достаточным для нового пользователя для использования программы должным образом, а также полный справочник со всеми вариантами и аргументами для искушенного пользователя.

Строки документации модуля должны, как правило, перечислять классы, исключения, функции (и любые другие объекты), которые экспортируются модулем, с краткими пояснениями (в одну строчку) каждого из них. (Эти строки, как правило, дают меньше деталей, чем первая строка документации к объекту). Строки документации пакета модулей (т.е. строка документации в __init__.py) также должны включать модули и подпакеты.

Строки документации функции или метода должны обобщить его поведение и документировать свои аргументы, возвращаемые значения, побочные эффекты, исключения, дополнительные аргументы, именованные аргументы, и ограничения на вызов функции.

Строки документации класса обобщают его поведение и перечисляют открытые методы и переменные экземпляра. Если класс предназначен для подклассов, и имеет дополнительный интерфейс для подклассов, этот интерфейс должен быть указан отдельно (в строке документации). Конструктор класса должен быть задокументирован в документации метода __init__. Отдельные методы должны иметь свои строки документации.

Если класс - подкласс другого класса, и его поведение в основном унаследовано от этого класса, строки документации должны отмечать это и обобщить различия.

Строки документации - строковые литералы, которые являются первым оператором в модуле, функции, классе или определении метода.

Все модули должны, как правило, иметь строки документации, и все функции, классы, экспортируемые модулем, также должны иметь строки документации. Публичные методы (в

том числе __init__) также должны иметь строки документации. Пакет модулей может быть документирован в __init__.py.

Существует две формы строк документации: однострочная и многострочная.

Однострочные строки документации должны умещаться на одной строке.

Раздаточный материал № 166

```
def rectangle():
"""Вычисление площади прямоугольника"""
a = float(input("Ширина %s: " % figure)) # обращение к глобальной
b = float(input("Высота %s: " % figure)) # переменной figure
print("Площадь: %.2f" % (a*b))
```

Многострочные строки документации

Многострочные строки документации состоят из однострочной строки документации с последующей пустой строкой, а затем более подробным описанием. Первая строка может быть использована автоматическими средствами индексации, поэтому важно, чтобы она находилась на одной строке и была отделена от остальной документации пустой строкой. Первая строка может быть на той же строке, где и открывающие кавычки, или на следующей строке. Вся документация должна иметь такой же отступ, как кавычки на первой строке.

Раздаточный материал № 167

```
def triangle():
"""Вычисление площади треугольника

    Используется общепринятая формула

    """
a = float(input("Основание %s: " % figure))
    h = float(input("Высота %s: " % figure))
print("Площадь: %.2f" % (0.5 * a * h))
```

При документировании фрагмента кода создается объект строкового типа, который сохранятся в атрибуте __doc__.

Кроме этого, содержимое строки документации отображается в всплывающей подсказке при наведении мыши на имя объекта в коде программы.

Для вывода в консоль содержимого атрибуте __doc__:

Раздаточный материал № 168

```
"""Это описание модуля"""

def rectangle():
"""Вычисление площади прямоугольника"""
pass

def triangle():
"""Вычисление площади треугольника

    Используется общепринятая формула
```

```
    """
pass

print(rectangle.__doc__)
print(triangle.__doc__)
```

Модульное программирование

Модульное программирование — это процесс разбиения большой и громоздкой задачи на отдельные, более маленькие, управляемые подзадачи и модули. Все это называется декомпозицией. Далее отдельные модули могут быть скомпонованы вместе, как строительные блоки, для создания более крупного приложения.

У модульного подхода при проектировании кода больших приложений есть сразу несколько преимуществ:

Простота: Вместо того, чтобы думать о всей проблеме в целом, обычно, в модуле фокусируются на решении одной, относительно небольшой, части программы. Работая над одним модулем, сужается область размышлений, что делает разработку проще и менее подверженной ошибкам.

Модифицируемость: Обычно, модули имеют логические границы между различными задачами проблемы в целом. Если в модулях свести к минимуму взаимозависимости, то снижается вероятность того, что модификации одного модуля окажут влияние на другие части программы. Возможно, вы даже сможете вносить изменения в модуль, не зная ничего о приложении, для которого он написан. Таким образом, над одним приложением может работать большая группа программистов, что есть совместная разработка.

Повторное использование кода: Функциональность, определенная в одном модуле, может быть легко использована повторно (через соответствующий интерфейс) другими приложениями, что избавляет от необходимости дублирования.

Область действия: Обычно, в модуле определяется отдельное пространство имен, что помогает избежать коллизий между идентификаторами в разных областях программы.

Модули

Модулем в языке Python называется любой файл с программным кодом. Каждый модуль может импортировать другой модуль, получая таким образом доступ к атрибутам (переменным, функциям и классам), объявленным внутри импортированного модуля.

Python есть три способа определения модуля:

- Модуль может быть написан на самом Python.
- Модуль может быть написан на C и динамически подгружен во время исполнения, как модуль re (regular expression).
- Модуль, встроенный в интерпретатор, как инструмент itertools.

Во всех случаях доступ к модулю предоставляется одинаково — с помощью оператора import.

Когда интерпретатор выполняет оператор import, то он ищет файл модуля в следующих каталогах в порядке приоритетности:

- Текущий каталог, т.е. тот каталог, из которого был запущен наш скрипт с оператором import.

- В списке каталогов, определенном в установленной переменной окружения PYTHONPATH.
- Каталоги стандартной библиотеки.
- Содержимое файлов с расширением pth.
- Подкаталог site-packages, где размещаются сторонние расширения.

В результате, в переменной окружения sys.path модуля sys содержится список каталогов для поиска импортируемого модуля:

Раздаточный материал № 169

```
>>>import sys
>>>sys.path
['C:\\Program Files\\JetBrains\\PyCharm Community Edition 2021.1.2\\plugins\\python-ce\\helpers\\pydev', 'C:\\Program Files\\JetBrains\\PyCharm Community Edition 2021.1.2\\plugins\\python-ce\\helpers\\third_party\\thriftpy', 'C:\\Program Files\\JetBrains\\PyCharm Community Edition 2021.1.2\\plugins\\python-ce\\helpers\\pydev',
'C:\\Users\\OLGA\\AppData\\Local\\Programs\\Python\\Python38-32\\python38.zip',
'C:\\Users\\OLGA\\AppData\\Local\\Programs\\Python\\Python38-32\\DLLs',
'C:\\Users\\OLGA\\AppData\\Local\\Programs\\Python\\Python38-32\\lib',
'C:\\Users\\OLGA\\AppData\\Local\\Programs\\Python\\Python38-32',
'C:\\Users\\OLGA\\AppData\\Local\\Programs\\Python\\Python38-32\\lib\\site-packages',
'C:\\PythonProjects\\zab', 'C:/PythonProjects/zab']
```

Поиск прекращается после первого найденного модуля. Таким образом, если в каталогах существуют одноименные модули, то будет использоваться модуль из папки, которая расположена первой в списке путей поиска.

Встроенная функция dir() возвращает список всех имен, определенных в пространстве. При задании в качестве аргумента имени модуля, dir() перечислит имена, определенные в этом модуле:

Раздаточный материал № 170

```
>>>import main
>>>dir(main)
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__', …]
```

Что бы можно было различать, когда файл загружается как модуль и когда он запускается как отдельный скрипт необходимо проанализировать значение предопределенного атрибута __name__. Для запускаемого модуля он содержит значение "__main__", а для импортируемого модуля — его имя.

Проверить, является модуль главной программой или импортированным модулем, позволяет код:

Раздаточный материал № 171

```
if __name__ == '__main__':
    print_hi('PyCharm')
```

Перезагрузка модуля

Из соображений эффективности модуль загружается только один раз за сеанс интерпретатора. Это хорошо для определений функций и классов, которые обычно составляют основную часть содержимого модуля. Но модуль также может содержать исполняемые операторы (обычно для инициализации) и они будут выполняться только при первом импорте модуля.

Раздаточный материал № 172

Существует файл mod.py:

```
a = [10, 20, 30]
print('a =', a)
```

```
>>>import mod
a = [100, 200, 300]
>>>import mod
>>>import mod
```

Оператор print() не выполняется при последующем импорте.

Если происходят изменения в модуле и его необходимо перезагрузить, то нужно либо перезапустить интерпретатор, либо использовать функцию с именем reload() из модуля importlib:

Раздаточный материал № 173

```
>>>import importlib
>>>importlib.reload(Doc.mod)
a =  [10, 20, 30]
<module 'Doc.mod' from 'C:\\PythonProjects\\zab\\Doc\\mod.py'>
```