3. Требования к коду

В 3.1 Какие документы устанавливают основные требования к коду Python?

Ответ: это документы PEP8, PEP257

В 3.2 Как именуются константы в Python и можно ли их изменять?

Ответ: константы в Python пишутся заглавными буквами, можно использовать нижнее подчеркивание. Менять их значение можно в процессе выполнения программы.

В 3.3 Как именуются переменные в Python?

Ответ: все буквы в названии переменной маленькие, можно использовать нижнее подчеркивание.

В 3.4 Как именуются функции в Python?

Ответ: пишутся с помощью маленьких букв, должны отражать, что делает функция.

В 3.5 Как именуются классы в Python?

Ответ: каждое слово начинается с заглавной буквы. Нижние подчеркивания не используются.

В 3.6 Какая максимальная длина строки должна быть согласно PEP8?

Ответ: 79 символов

В 3.7 Порядок импортов и их сортировка?

Ответ: вначале импортируются модули стандартной библиотеки Python, потом модули скачанных библиотек, потом свои собственные модули. Они отделяются друг от друга строкой.
Импортируется чаще всего через from, импортируемые компоненты должны быть в алфавитном порядке.

В 3.8 Можно ли применять бексплеши для переноса?

Ответ: нельзя

В 3.9 Какие кавычки должны быть в программе?

Ответ: двойные или одиночные, но везде только одного из этих двух типов.

В 3.10 Стоит ли всегда использовать ключевое слово else в условной конструкции?

Ответ: если else можно заменить на GuardBlock, то лучше так и поступить

В 3.11 Какой конструкцией должен быть закрыт код в исполняемом ру-файле?

Ответ: конструкцией if __name__ == “__main__”

В 3.12 Если в программе возможен выбор между списками и кортежами, что предпочтительнее выбирать?

Ответ: кортежи

В 3.13 Какая специфика использования f-строк?

Ответ: применяется только подстановка переменных и нет логических или арифметических операций, вызов функций подобной динамики.

В 3.14 Как должны быть названы переменные?

Ответ: переменные названы в соответствии с их смыслом, по-английски, нет однобуквенных названий и транслита. В названии переменной не должен содержаться ее тип. При необходимости применять аннотацию типов.

В 3.15 О чем документ PEP257?

Ответ: о документированных строках – Docstrings

В 3.16 Что такое документированная строка?

Ответ: Документационная строка — это строковый литерал, являющийся первой инструкцией в определении модуля, функции, класса или метода. Такая строка становится доступна при обращении с специальному атрибуту _ _ doc _ _ этого объекта.

В 3.17 Для каких объектов применяется Docstrings?

Ответ: Docstring применяется для всех функций, классов и библиотек.

В 3.18 Расскажите основные правила документирования?

Ответ: Для согласованности всегда используйте """тройные двойные кавычки""" вокруг документационной строки. Документационная строка — это «фраза», заканчивающаяся точкой. Если весь docstring не помещается в строку, вы можете вынести закрывающие кавычки на отдельную линию.

В 3.19 Стоит ли документировать каждую строчку кода?

Ответ: Комментировать каждую строчку кода считается плохим тоном. Используйте комментарии, когда нужно: указать на участок кода, на который стоит обратить внимание; пояснить сложные алгоритмы или логику; указать на код, который нужно доработать; указать на код, который хочется позже разобрать.

В 3.20 Какие существуют инструменты для Python для проверки кода на соответствие стандартам?

Ответ: используют линтеры - в программировании линтерами принято называть инструменты для анализа кода, которые помогают находить места, где код не соответствует указанному стандарту.

В 3.21 Назовите несколько примеров инструментов для Python для проверки кода на соответствие стандартам?

Ответ: flake8 – для PEP8, mypy – для аннотации типов

В 3.22 Что такое аннотация типов и для чего она нужна?

Ответ: Чтобы держать типизацию под контролем — применяют **аннотации типов данных** (_Type Hints_, дословный перевод с английского — «подсказки типов»). Python не оставляет аннотации совсем без внимания: он считывает _Type Hints_ и сохраняет их в словарь `__annotations__`. Содержимое этого словаря можно вывести на экран:

print(_ _ annotations _ _ _) >>> {'name': <class 'str'>, 'var_for_bool': <class 'bool'>}

В 3.23 Какая библиотека существует для аннотации типов и как она работает?

Ответ: Библиотека (модуль) typing нужна для аннотации типов в коде программы, чтобы вы и другие программисты, которые читают ваш код, понимали, что принимает и отдает та или иная функция. Аргумент функции принимает строки или числа, а переменная может содержать число или None. В таких случаях для аннотирования типов данных применяются компоновщики.

С помощью компоновщика можно указать для переменной несколько возможных типов данных.

В 3.24 Что анномирует типы даннаых, которые могут содержать 2 типа данных?

Ответ: Метод Optional

``from typing import Optional

``text: Optional[str] - переменная text ожидает str или None

``text = None - проблем нет

В 3.25 Что используется, если переменная должна принимать данные нескольких разных типов?

Ответ: применяют аннотацию Union

``from typing import Union``

``def hundreds(x: Union[int, str]) -> str:

    ``return str(x * 100)``

``hundreds(100)

``chundreds('сто')

В 3.26 Что применять, когда не нужно ограничивать возможные типы переменной?

Ответ: Метод Any – но лучше его никогда не применять

``x: Any

``x = 12210

``x = 'Строка'

``x = True

``x = None

В 3.27 Как производится аннотация коллекций?

Ответ: Если используется версия Python 3.9+ тогда имортировать typing не нужно, а сразу без импорта записать dict, list, tuple, set. Если версия питона ниже есть from typing import Sequence, Dict, List, Tuple, Set

---

В 3.28 Что использовать если функция передаётся в качестве аргумента в другую функцию или в метод и там вызывается?

Ответ: такому аргументу присваивается тип Callable.

``from typing import Callable

``def printer() -> None:

    ``print("Вызови меня!")

``def returner(word: str) -> str:

    ``return word

``def app(printed_inside: Callable[[], None], returned_inside: Callable[[str], str]) -> None:

    ``printed_inside()

    ``print(returned_inside('Нет, вызови меня!'))

``app(printer, returner)

``app(printer, printer)