Python функции

План

0. Источники

1. Принцип DRY

2. Создание функции

3. “Хорошая” Функция

4. Блок **if \_\_name\_\_ == "\_\_main\_\_"**

5. Пустая функция (Code)

6. Функции с позиционними оргументами (Code)

7. Функции с именованными (ключевими) аргументами (Code)

8. Функции с позиционними и именованными єлементами (Code)

9. Использование структури \*args, \*\*kwargs (Code)

10. Особенность работи инструкции return (Code)

11. Область видимость и глобальные переменные (Code)

12. Анонимные функции, инструкция lambda (Code)

0. Источники.

!!!!!!!!!!!!!!!!!!!!!

<https://tproger.ru/translations/python-built-ins-worth-learning/>

https://pythonworld.ru/osnovy/vstroennye-funkcii.html

!!!!!!!!!!!!!!!!!!!!!

https://python-scripts.com/functions-python

<https://pythonru.com/osnovy/funkcii-v-python>

https://habr.com/ru/company/piter/blog/426381/

1. Принцип DRY

2. Создание функции

Функция в python - объект, принимающий аргументы и возвращающий значение. Обычно функция определяется с помощью инструкции **def**.

Инструкция **return** говорит, что нужно вернуть значение.

Говоря о функциях в Python, нужно упомянуть два аспекта:

1) функция в Python — есть объект специального вида, который можно передавать в виде аргумента другим функциям;

2) внутри функций в Python вы можете создавать другие функции, а также вызывать их, возвращая результат посредством return.

В языке Python определение функции должно предшествовать ее вызовам. Это связано с тем, что интерпретатор читает код строка за строкой и о том, что находится ниже по течению, ему еще неизвестно. Поэтому если вызов функции предшествует ее определению, то возникает ошибка (выбрасывается исключение NameError)

Польза функций не только в возможности многократного вызова одного и того же кода из разных мест программы. Не менее важно, что благодаря им программа обретает истинную структуру. Функции как бы разделяют ее на обособленные части, каждая из которых выполняет свою конкретную задачу.

Она внятно названа

3. “Хорошая” Функция

https://habr.com/ru/company/piter/blog/426381/

Функцию Python «хорошей», если она удовлетворяет *большинству* пунктов из следующего списка (выполнить все пункты для конкретной функции порой невозможно):

1. Она внятно названа
2. Соответствует принципу единственной обязанности
3. Содержит докстроку
4. Возвращает значение
5. Состоит не более чем из 50 строк
6. Она идемпотентная и, если это возможно, чистая

1.

Соблюдать нейм конвеншн проекта команди

Лучше использовать полные английские слова, а не аббревиатуры и не малоизвестные сокращения.

2.

Согласно этому принципу (в нашем случае) у функции должна быть единственная ответственность. То есть, она должна делать одну и только одну вещь. Один из самых веских доводов в пользу этого: если функция делает всего одну вещь, то и переписывать ее придется в единственном случае: если эту самую вещь придется делать по-новому

Такое разделение также серьезно упрощает тестирование функции, а еще позволяет не только разбить ее на функции в рамках одного и того же модуля, но даже разнести функции в совершенно разные модули, если это уместно. Это дополнительно способствует более чистому тестированию и упрощает поддержку кода.

3.

PEP 257 -- Docstring Conventions

https://www.python.org/dev/peps/pep-0257/

* Для каждой функции нужна докстрока
* В ней следует соблюдать грамматику и пунктуацию; писать законченными предложениями
* Докстрока начинается с краткого (в одно предложение) описания того, что делает функция
* Докстрока формулируется в предписывающем, а не в описательном стиле

4.

Если функция даже не станет возвращать значения, то интерпретатор Python «принудит» ее возвращать None.

5.

**Длина функции прямо сказывается на ее удобочитаемости и, следовательно, на поддержке**. Поэтому старайтесь, чтобы ваши функции оставались короткими.

6.

Идемпотентная функция при одинаковом наборе аргументов всегда возвращает одно и то же значение, независимо от того, сколько раз ее вызывают.

Тестируемость и удобство в поддержке.

Рефакторинг (изменение структури без изменения поведения) при работе с идемпотентными функциями – вообще легкая прогулка.

В функциональном программировании функция считается чистой, если она, *во-первых*, идемпотентна, а *во-вторых* – не вызывает наблюдаемых **побочных эффектов**.

В функциональном программировании функция считается чистой, если она, *во-первых*, идемпотентна, а *во-вторых* – не вызывает наблюдаемых **побочных эффектов**.

Чистая функция не оказывает побочных эффектов. Она не только не использует никаких «внешних данных» при расчете значения, но и не взаимодействует с остальной программой/системой, только вычисляет и возвращает указанное значение. В чистых функциях нет инструкций логирования или вызовов print(). При работе они не обращаются к базе данных и не используют соединений с интернетом. Не обращаются к нелокальным переменным и не изменяют их. **И не вызывают других не-чистых функций**.

4. Блок **if \_\_name\_\_ == "\_\_main\_\_"**

<https://pyneng.readthedocs.io/ru/latest/book/11_modules/if_name_main.html>

[https://ru.stackoverflow.com/questions/515852/%D0%A7%D1%82%D0%BE-%D0%B4%D0%B5%D0%BB%D0%B0%D1%8E%D1%82-if-name-main](https://ru.stackoverflow.com/questions/515852/Что-делают-if-name-main)

[https://coderoad.ru/419163/%D0%A7%D1%82%D0%BE-%D0%B4%D0%B5%D0%BB%D0%B0%D0%B5%D1%82-if-\_\_name\_\_-\_\_main\_\_-do](https://coderoad.ru/419163/Что-делает-if-__name__-__main__-do)

5. Пустая функция (Code)

6. Функции с позиционними оргументами (Code)

7. Функции с именованными (ключевими) аргументами (Code)

8. Функции с позиционними и именованными єлементами (Code)

9. Использование структури \*args, \*\*kwargs (Code)

10. Особенность работи инструкции return (Code)

Инструкциия return преривает дальнейшую работу функции

11. Область видимость и глобальные переменные (Code)

В *Python* выделяют четыре области видимости для переменных. (Интервью LEGB)

* Local (С) (Code)

Локальные создаются внутри методов, функций и прочих блоках кода, при этом, после выхода из такого блока переменная удаляется из памяти.

* Enclosing (Code)
* Global (Code)

Глобальные существует в течении всего времени выполнения программы;

* Built-in

Проблема глобальной переменной (Code)

* Становится трудно сказать, когда и где переменная была определена.
* Когда мы определяем «а» как глобальную в одном месте, мы можем случайно переопределить её значение в другом, что может вызвать логическую ошибку, которую не просто исправить.

## 12. Анонимные функции, инструкция lambda (Code)