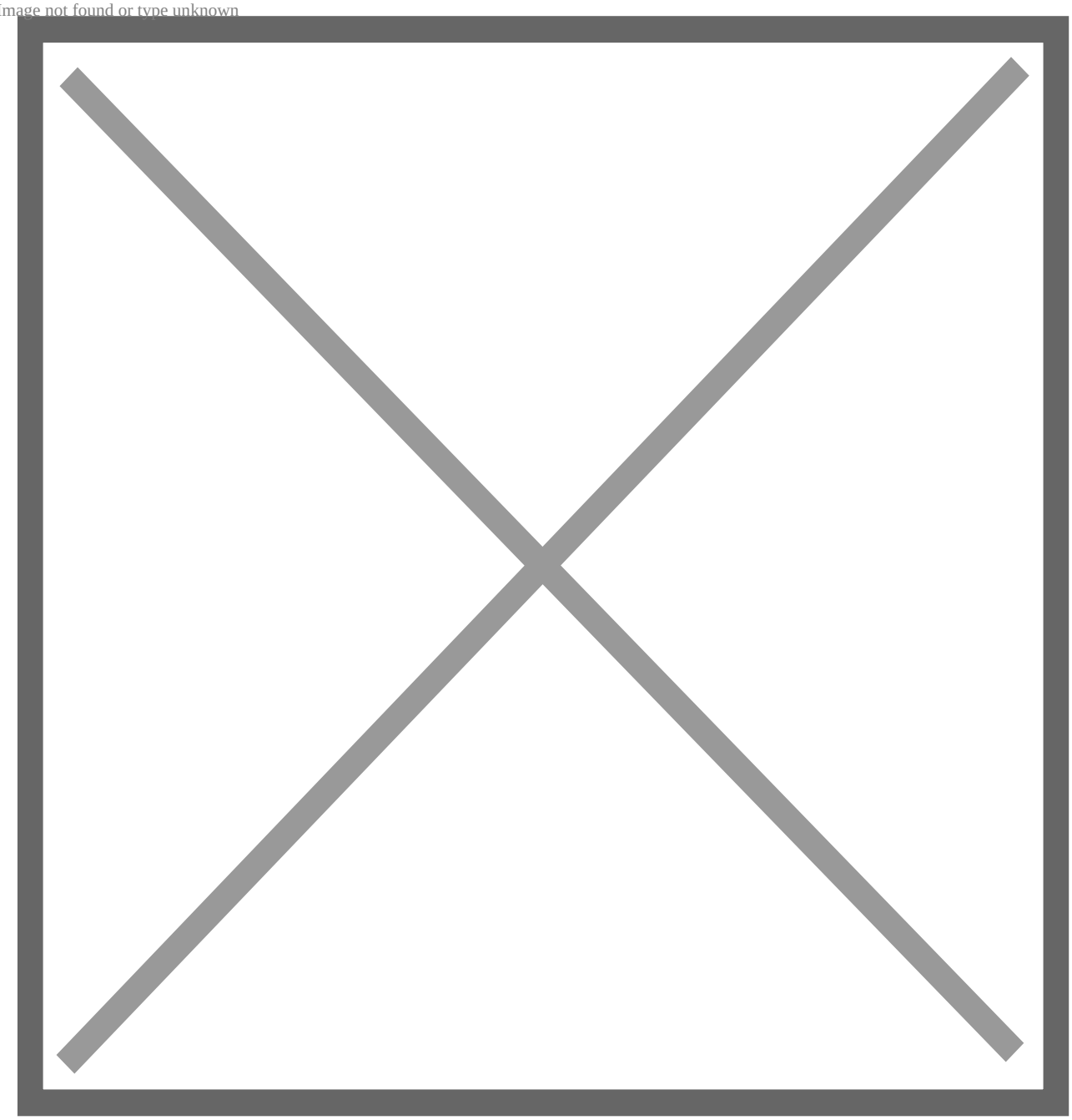
Понимание функции Sum в Python [с примерами]

Описание

Узнайте все о **функции sum()** в Python: от синтаксиса до использования ее с различными итерациями – с полезными примерами кода. При работе с итеративными переменными Python, такими как список чисел, часто требуется найти сумму всех элементов списка. Мы будем сталкиваться с подобными операциями и при работе с другими итерациями, такими как кортежи и множества. Это можно сделать несколькими различными способами, но рекомендуемый Питоном способ – использование встроенной функции sum(). Здесь мы начнем с рассмотрения других подходов, таких как цикл и определение функции. Затем мы перейдем к изучению синтаксиса функции sum() в Python и примеров кода для ее лучшего понимания.

Суммирование значений в итерабельной таблице Python

Image not found or type unknown



Рассмотрим следующий список чисел:

```
>>> nums = [2,8,5,3,11,7,9]
```

Наша цель – найти сумму всех чисел в списке. Мы скоро перейдем к функции Python sum(), но начнем с некоторых других подходов, которые мы можем использовать. К ним относятся:

- Использование простого цикла for

- Использование функции reduce() из модуля functools
- Определение пользовательской функции

Использование циклов

Чтобы найти сумму всех элементов списка, мы можем использовать цикл for следующим образом:

- Инициализируйте переменную total нулем.
- Пройдитесь по списку nums и получите доступ к каждому числу.
- Добавьте число к сумме.

```
>>> nums = [2,8,5,3,11,7,9]
>>> total = 0
>>> for num in nums:
...     total += num
...
>>> total
```

```
45
```

Использование функции reduce

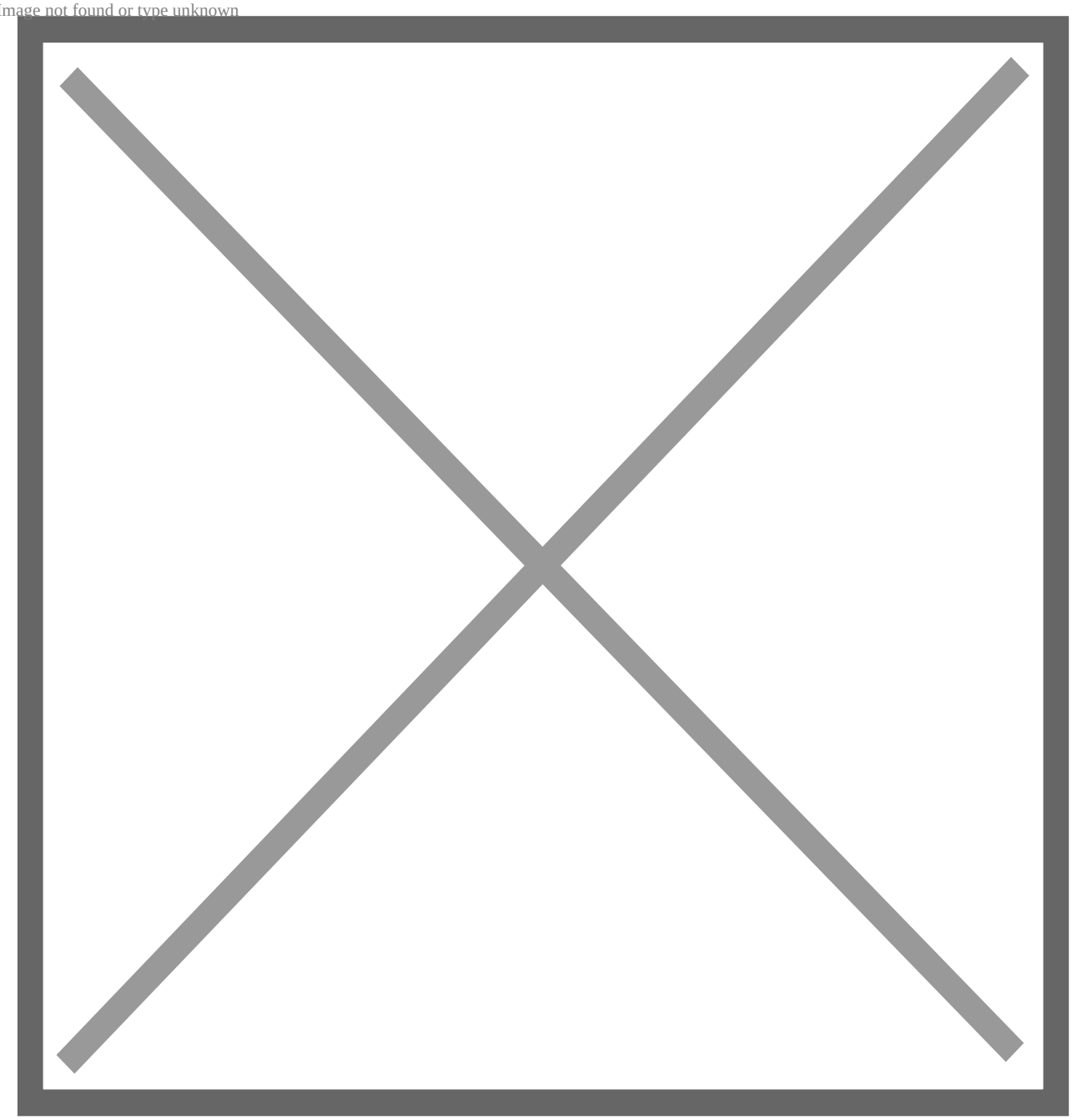
Другой подход к суммированию итерабельных таблиц – использование функции reduce(). Функция reduce, встроенная в модуль Python functools, принимает функцию и итерируемое число. И уменьшает итерируемое число, последовательно применяя функцию к элементам итерируемого числа. Здесь мы используем лямбда-функцию для определения сложения двух чисел и передаем список nums в качестве итерабельной функции.

```
>>> nums = [2,8,5,3,11,7,9]
>>> from functools import reduce
>>> total = reduce(lambda n1, n2: n1 + n2, nums)
>>> total
```

```
45
```

Функция reduce() работает путем последовательного сложения двух чисел – слева направо – до тех пор, пока они не сведутся к одному значению суммы:

Image not found or type unknown



Использование пользовательской функции

Для этого мы также можем определить пользовательскую функцию. Здесь мы определяем функцию sum_list, которая:

- Принимает в качестве аргумента список чисел и
- Возвращает сумму элементов списка.

В теле функции используется конструкция цикла, которую мы рассматривали

ранее. Но определение функции дает нам возможность повторного использования.

```
>>> def sum_list(some_list):
...     total = 0
...     for num in some_list:
...        total += num
...     return total

...
```

Вызов функции sum_list() с числами в качестве аргументов возвращает сумму 45:

```
>>> nums = [2,8,5,3,11,7,9]
>>> total = sum_list(nums)
>>> total

45
```

Далее давайте познакомимся со встроенной функцией sum(). Она не только лаконична, но и надежна, поскольку хорошо работает с несколькими итерациями и типами данных.

Синтаксис функции sum в Python

Синтаксис для использования функции **sum()** следующий:

```
sum(iterable, start)
```

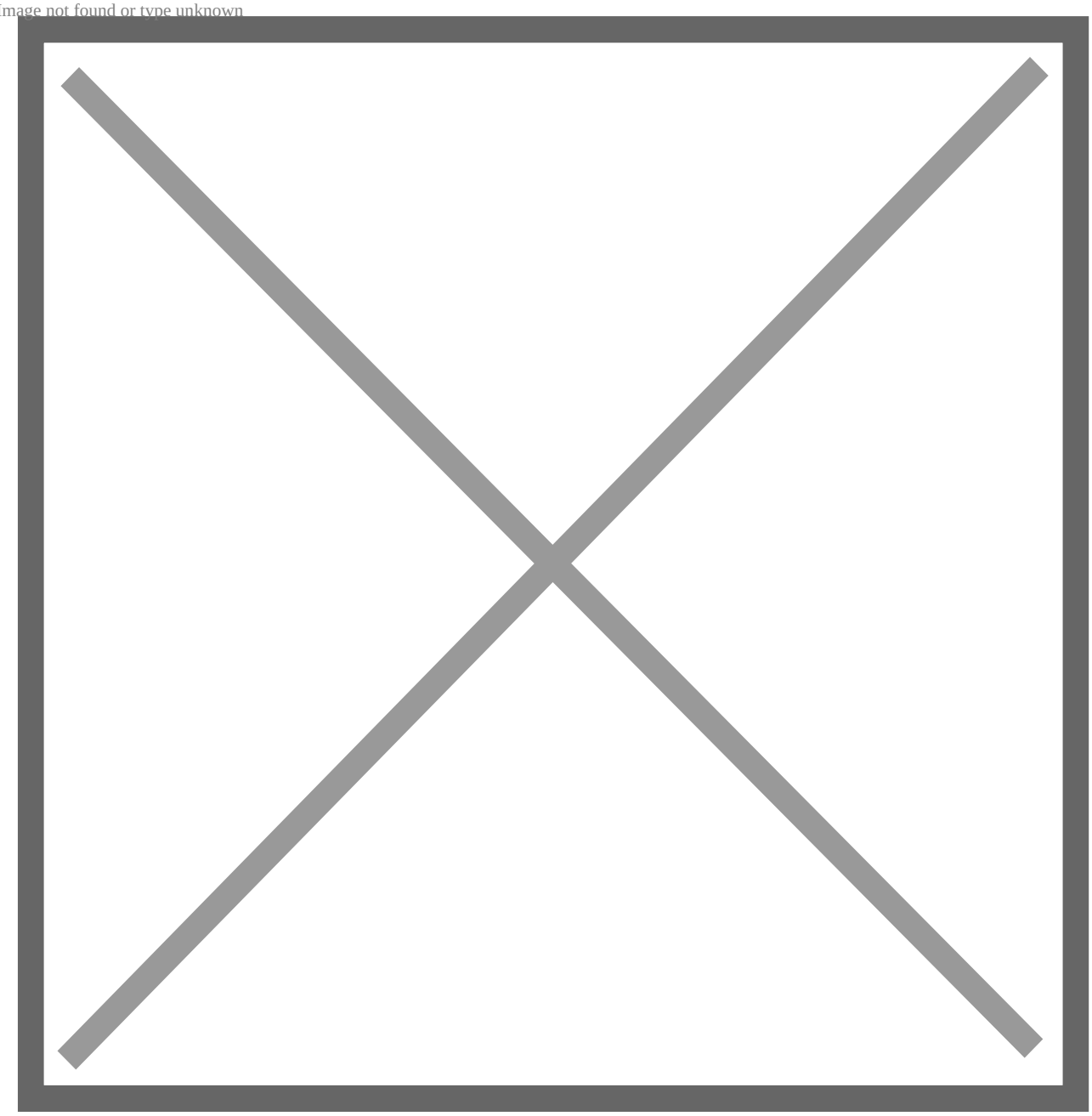
Здесь,

- **iterable** является обязательным аргументом. Это может быть любой итератор, для которого допустима операция суммирования, например, список или кортежи чисел. Вызов функции **sum()** со строками Python вызывает исключение TypeError (подробнее об этом позже).
- **start** – необязательный аргумент. Часто это числовое значение, которое добавляется к вычисленной сумме. Это может быть полезно, когда нужно добавить к результату постоянное значение.

Теперь, когда мы изучили синтаксис функции Python sum(), давайте используем ее для суммирования итераций.

Суммирование итерационных таблиц с помощью функции Sum

Image not found or type unknown



Список

Найдем сумму чисел в списке nums с помощью функции sum():

```
>>> nums = [2,8,5,3,11,7,9]
>>> sum_1 = sum(nums)
```

```
>>> sum_1
45
```

Использование необязательного начального значения

Чтобы добавить постоянное значение к сумме, мы можем использовать функцию sum() с необязательным начальным значением. Здесь мы передаем начальное значение 100 в качестве позиционного аргумента:

```
>>> sum_start = sum(nums,100)
>>> sum_start

145
```

Начальное значение также может быть указано в качестве аргумента ключевого слова:

```
>>> sum_start = sum(nums,start=10)
>>> sum_start

55
```

Кортеж

Функция sum() также работает с кортежами. Мы создаем кортеж nums_tuple путем приведения списка nums к кортежу:

```
>>> nums_tuple = tuple(nums)
>>> nums_tuple

(2, 8, 5, 3, 11, 7, 9)

>>> sum_2 = sum(nums_tuple)
>>> sum_2

45
```

Набор

Мы также можем использовать функцию sum() с набором чисел:

```
>>> nums_set = set(nums)
>>> nums_set
{2, 3, 5, 7, 8, 9, 11}
```

Здесь мы приводим список nums к множеству Python и вычисляем сумму элементовв nums_set.

```
>>> sum_3 = sum(nums_set)
>>> sum_3
```

45

Словарь

Рассмотрим следующий словарь student_dict с числовыми ключами. Обратите внимание, что происходит, когда вы вызываете функцию sum() с этим словарем в качестве аргумента.

```
>>> students_dict = {1:106,2:112,3:127}
>>> sum_4 = sum(students_dict)
>>> sum_4
```

6

Функция sum() по умолчанию возвращает сумму ключей.

Суммирование ключей

Мы знаем, что по умолчанию суммируются ключи словаря. Однако вы можете сделать это более явным, используя метод словаря keys() для доступа к ключам. А затем передать список ключей в функцию sum():

```
>>> sum_keys = sum(students_dict.keys())
>>> sum_keys
```

6

Суммирование значений

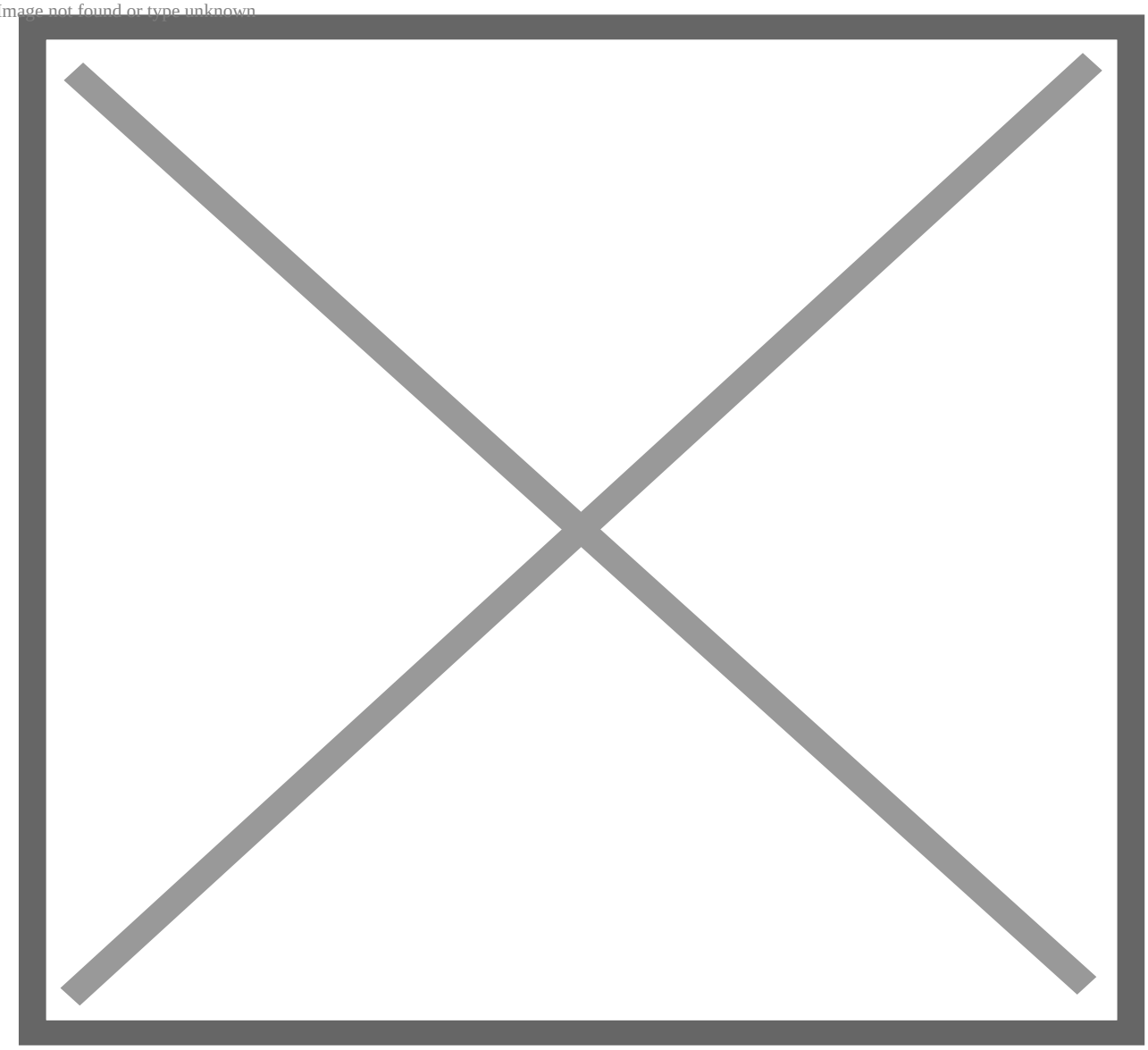
Если вы хотите просуммировать значения словаря, обратитесь к ним, вызвав метод values() объекта словаря:

```
>>> sum_vals = sum(students_dict.values())
>>> sum_vals
345
```

Использование функции Sum в Python с другими

числовыми типами данных

Image not found or type unknown



До сих пор мы видели, как использовать функцию sum() с итерациями целых чисел. Теперь давайте рассмотрим несколько примеров с другими числовыми типами данных.

Комплексные числа

Функция sum() также может быть использована для суммирования комплексных чисел. В этом примере nums_c – это список комплексных чисел:

```
>>> nums_c = [3 + 4j, 1 + 2j]
>>> sum_c = sum(nums_c)
>>> sum_c
```

```
(4+6j)
```

Числа с плавающей запятой

Здесь мы используем функцию sum() для суммирования списка чисел с плавающей точкой nums_f:

```
>>> nums_f = [1.8,2.5,3.6,7.2]
>>> sum_f = sum(nums_f)
>>> sum_f

15.100000000000001
```

Для повышения точности результата сложения чисел с плавающей точкой можно использовать функцию fsum() из модуля math для суммирования итераций со значениями с плавающей точкой.

Сплющивание с помощью функции Sum

Теперь давайте посмотрим, как функция sum() может быть использована для сплющивания и объединения итераций.

Сплющивание списка

Предположим, у нас есть вложенный список:

```
>>> lists = [[2,4,6],[3,5,7]]
```

Когда мы вызываем функцию sum(), передавая этот вложенный список в качестве аргумента вместе с пустым списком в качестве начального значения:

```
>>> sum(lists,[])
[2, 4, 6, 3, 5, 7]
```

Мы видим, что вложенный список теперь сплющился в один список чисел. Эквивалентно, если представить список в виде l3 = [l1,l2], функция sum() объединяет два списка l1 и l2, вложенных в список l3. В качестве небольшого упражнения попробуйте использовать функцию sum() для других вложенных итераций.

Распространенная ошибка: Не используйте

функцию sum() Python со строками

Поскольку мы видели, что функция sum() может использоваться для сплющивания и объединения списков (и других итераций, например, кортежей), очень соблазнительно подумать, что ее можно использовать и для объединения строк.

Но если вы попытаетесь это сделать, то столкнетесь с ошибкой TypeError:

```
>>> sum(['a','b','c'],'')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>

TypeError: sum() can't sum strings [use ''.join(seq) instead]
```

Поэтому функцию sum() нельзя использовать для суммирования (или конкатенации) строк. Однако, как видно из сообщения об ошибке выше, вы можете использовать метод join() для объединения списка строк в одну строку.

```
>>> ''.join(['a','b','c'])

'abc'
```

Заключение

В этом уроке мы узнали, как использовать встроенную функцию sum() для нахождения суммы всех элементов итерабельной таблицы. Общий синтаксис для использования функции sum() следующий: sum(iterable, start), где iterable – обязательный аргумент, а start – необязательный аргумент. Затем мы написали несколько примеров, чтобы понять, как использовать функцию sum() с такими итерациями, как списки, кортежи, множества и словари. Позже мы рассмотрели, как функция sum() может использоваться для сглаживания и конкатенации итераций – за исключением строк Python. Надеюсь, вы нашли этот учебник полезным.

Дата Создания
16.06.2023