Самое важное

Форматирование строк: format и f-strings:
В Python есть широкий набор инструментов, который позволяет выполнять над строками огромное множество изменений.

Один из методов — это метод format. С его помощью, например, можно вставить в строку какие-либо слова:

- username = input(‘Введите своё имя’) — получаем какое-либо слово;
- ‘Hello {name}!’.format(name=username) — подставляем его в строку при помощи метода format.

Параметром в нашем случае будет название, которое мы указали в самой строке в фигурных скобках, — {name}.

f-strings является «новым» способом форматирования строк. По сути, это аналог метода .format(), который работает немного быстрее, но при этом уступает format’у в красоте представления.

Для использования f-строк достаточно поставить перед строкой ‘f’:
username = input(‘Введите своё имя’)
print(f‘Hello {username}!’) — и в этом случае мы не создаем промежуточного параметра вроде name, который был ранее, мы сразу указываем название переменной, которую нужно использовать для подстановки.

Форматирование чисел при подстановках в строки
Помимо простой подстановки, мы можем уточнять какие-либо характеристики подставляемых элементов. Например, можем указать разделитель для больших целых чисел или указать количество знаков после запятой для вещественных чисел:
price = 23.594233
details_num = 500000
percent = 0.05

print("На складе осталось {:,d}".format(details_num)) — ‘:’ объявляет подстановку, ‘d’ говорит о том, что работа идёт с целыми числами, а ‘,’ указывает методу, чем нужно разделять большое целое число.
В итоге мы получим:
На складе осталось 500,000.

print("Каждая деталь стоит {:.2f}".format(price)) — здесь ‘.2’ говорит о количестве знаков после точки, а ‘f’ говорит о том, что мы работаем с вещественными числами.
В итоге мы получим:
Каждая деталь стоит 23.59.

print("Перевод в проценты числа 0.05 будет выглядеть так {:.1%}".format(percent)) — в этом примере мы переводим число в проценты, указывая при этом количество знаков после точки, которое хотим получить в итоге.
Результатом будет:
Перевод в проценты числа 0.05 будет выглядеть так 5.00%.

print("Большие числа иногда удобнее записать в экспоненциальном виде {:.0e}".format(details_num)) — вместо записи с разделителем мы можем также записать большое число через экспоненциальный вид:
Большие числа иногда удобнее записать в экспоненциальном виде 5e+05.

Методы split и join
Метод split, как видно уже по названию, создан для того, чтобы разделять строку.

Синтаксис
str.split(sep=None, maxsplit=-1)
<строка, которую хотим разделить>.split(<символы, по которым будет идти разделение>, <максимальное ограничение разделений>)

Пример
test_for_split = 'Мы хотим получить каждый элемент отдельно!'
print(test_for_split.split()) — запустив метод без параметров, мы позволяем Python использовать стандартные параметры метода. По умолчанию разделителем является пробел, а максимальное ограничение равно -1, то есть количество разделений не ограничено.

В итоге мы получим СПИСОК: ['Мы', 'хотим', 'получить', 'каждый', 'элемент', 'отдельно!'].
Python проходил по строке и искал пробелы. Найдя пробел, он выделял область от начала до этого пробела (не включая сам пробел!) в отдельное слово.
После отделения Python продолжает поиск до следующего пробела. В итоге мы получаем список со всеми словами, которые были разделены пробелами.

test_for_split = 'Мы,хотим,получить каждый,элемент отдельно!'
print(test_for_split.split(‘,’)) — при этом мы можем использовать и любой другой строчный символ (или символы), чтобы по ним разделять начальную строку.
Если мы используем на новой строке разделение по запятым, то получим:
['Мы', 'хотим', 'получить каждый', 'элемент отдельно!'].

А если мы укажем ограничение количества разделений:
print(test_for_split.split(',', 1)), то получим и вовсе:
['Мы', 'хотим,получить каждый,элемент отдельно!'].

Метод join работает наоборот. Он нужен для соединения элементов в строку. При этом применяется метод join к разделителю, которым вы хотите разделить между собой элементы:

```
test_for_join = ['Мы', 'хотим', 'получить', 'каждый', 'элемент', 'отдельно!']
print(','.join(test_for_join))
```

При split мы обращались к строке и в качестве параметра использовали разделитель. При join мы обращаемся к разделителю и в качестве параметра используем список. Из списка с разделенными элементами после использования join мы получим строку: «Мы,хотим,получить,каждый,элемент,отдельно!».

Как и со split, мы можем использовать любой разделитель, который только пожелаем (пока он остаётся строкой):

```
test_for_join = ['Мы', 'хотим', 'получить', 'каждый', 'элемент', 'отдельно!']
print('- даже так? - да - '.join(test_for_join))
```

Теперь вместо разделителя мы используем строку «— даже так? — да —», и она будет подставлена между всеми элементами, которые мы объединяем:
Мы— даже так? — да — хотим— даже так? — да — получить— даже так? — да — каждый— даже так? — да — элемент— даже так? — да — отдельно!

Синтаксис
str.join(iterable)
В качестве str мы подставляем разделитель, в качестве iterable — итерируемый объект с элементами в виде строк.

Методы startswith, endswith
Эти методы схожи по своей логике:

- startswith возвращает True, если проверяемая строка начинается с указанной вами подстроки, иначе возвращает False;
- endswith возвращает True, если проверяемая строка заканчивается на указанную вами подстроку, иначе возвращается False.

```
start_end = 'Начало, конец'
print(start_end.startswith('Начало')) — вернёт True
print(start_end.endswith('конец')) — вернёт True
```

Важно также понимать, что такой поиск чувствителен к регистру:
print(start_end.startswith('начало')) — вернёт False
print(start_end.endswith('Конец')) — вернёт False

Также вы можете указывать не один вариант, а, например, кортеж вариантов, каждый из которых будет проверен на наличие в вашей строке. Пример подобных приёмов можно найти в документации.

Методы upper, lower

Эти два метода позволяют привести все буквы в вашей строке к верхнему/нижнему регистру:
start_end = 'Начало, конец'
print(start_end.upper()) → НАЧАЛО, КОНЕЦ
print(start_end.lower()) → начало, конец

Иногда эти методы могут помочь при проверке слов. Например, мы ждём на вход слово «Привет», но пользователь случайно вводит «ПрИвЕт», и наша проверка if user_input == “Привет”: уже не сработает.

Однако если мы напишем if user_input.lower() == ‘привет’:, то мы сразу учтём все возможные варианты из больших/маленьких букв слова «привет».

Не допускай следующих ошибок!

Не забывайте, что join работает только со строками!
test_for_join = [1, '3', 3]
print('-'.join(test_for_join)) — такой вызов обернётся ошибкой, ведь в списке у нас содержатся числа и строки, а не только строки.