СОДЕРЖАНИЕ

Строковый тип данных

Строка (тип данных **str**) — это последовательность символов, заключённая в одинарные, двойные или тройные кавычки. Строки в Python являются неизменяемыми, что значит, что после создания строка не может быть изменена — любые операции создают новую строку, а не изменяют исходную. Это нужно помнить при работе со строками, чтобы избежать ошибок.

Например, считывание строки с клавиатуры осуществляется так:

```
name = input()
```

Распространённая ошибка — писать s = str(input()). Команда input() уже возвращает строку, и добавление функции str() избыточно. Это всё равно, что заказывать напиток, который уже стоит перед вами — тавтология.

Порядок символов в строке и индексы

Каждый символ в строке имеет свой порядковый номер, называемый **индексом**. В Python нумерация индексов начинается с **нуля**. Например, в строке "Python" символ P находится на нулевом индексе, y на первом, и так далее:

```
word = "Python"
print(word[0])  # Вывод: P
print(word[1])  # Вывод: y
```

Также в Python существуют **отрицательные индексы**, с помощью которых можно обращаться к символам с конца строки. Например:

```
print(word[-1])  # Вывод: n
print(word[-2])  # Вывод: o
```

Однако, если указать индекс, который выходит за пределы строки, программа выдаст ошибку:

```
s = "hello"
print(s[5])  # Ошибка IndexError
```

Здесь индекс 5 недоступен, потому что последний символ строки имеет индекс 4. В случае, если индекс выходит за пределы допустимого диапазона, Python выбрасывает ошибку IndexError, предупреждая о неверном обращении к символу строки.

Срезы строк

Срезы позволяют извлекать **подстроки** из строки. Синтаксис срезов выглядит так:

```
string[start:end:step]
```

- start — индекс, с которого начинается срез.
- end — индекс символа, до которого идёт срез (не включительно).
- step — шаг, с которым выполняется срез (по умолчанию шаг равен 1).

Некоторые из этих параметров можно не указывать:

- Если не указан start, срез начинается с начала строки.
- Если не указан end, срез продолжается до конца строки.
- Если не указан step, срез выполняется с шагом 1.

```
word = "programming"
print(word[0:5])    # Вывод: progr
print(word[5:])     # Вывод: amming (срез до конца строки)
print(word[:5])     # Вывод: progr (срез с начала строки)
print(word[::2])    # Вывод: pormig (с шагом 2)
```

Важно: если не указан шаг среза, начальный индекс должен быть меньше конечного, иначе срез будет пустым. Пример работы среза с отрицательным шагом (для развертывания строки):

```
print(word[::-1])  # Вывод: gnimmargorp
```

Длина строки

Для того чтобы узнать длину строки, используется встроенная функция **len()**:

```
word = "hello"
print(len(word))  # Вывод: 5
```

Методы строк

Методы строк — это функции, которые позволяют выполнять различные операции со строками. Рассмотрим некоторые из них:

Название метода	Описание	Пример работы
str.upper()	Преобразует строку в верхний регистр	"python".upper() → 'PYTHON'
str.lower()	Преобразует строку в нижний регистр	"PYTHON".lower() → 'python'
str.isupper()	Проверяет, все ли символы в верхнем регистре	"HELLO".isupper() → True

str.islower()	Проверяет, все ли символы в нижнем регистре	"hello".islower() → True
str.isdigit()	Проверяет, состоит ли строка из цифр	"12345".isdigit() → True
str.capitalize()	Делает первую букву заглавной	"hello".capitalize() → 'Hello'
str.title()	Первые буквы всех слов делает заглавными	"hello world".title() → 'Hello World'
str.find(sub)	Возвращает индекс первого вхождения подстроки	"hello".find('l') → 2
str.rfind(sub)	Возвращает индекс последнего вхождения подстроки	"hello".rfind('l') → 3
str.count(sub)	Возвращает количество вхождений подстроки	"hello".count('l') → 2

Методы find() и rfind() в Python возвращают -1, если подстрока не найдена. Пример для метода find():

```
text = "hello"
index = text.find("z")  # Пытаемся найти символ, которого нет в строке
print(index)  # Вывод: -1
```

Методы строк позволяют эффективно выполнять различные преобразования текста и упрощают работу с ними.

При вызове методов у строки пробелы вокруг точки и аргументов не ставятся:

```
word.upper()  # правильно
word . upper ()  # ошибка оформления
```

Таким образом, работа со строками в Python предоставляет мощные инструменты для их обработки, начиная от доступа к отдельным символам и заканчивая сложными преобразованиями с помощью методов.

Конкатенация строк

Конкатенация — это процесс объединения строк. В Python для этого используется оператор сложения (+). Это означает, что можно "складывать" строки вместе, создавая одну длинную строку:

```
first_name = "John"
last_name = "Doe"
full_name = first_name + " " + last_name
print(full_name)  # John Doe
```

Использование конкатенации полезно в случаях, когда нужно создать одну строку из нескольких частей. Например, при выводе сообщения:

```
name = "Alice"
print("Hello, " + name + "!")  # Hello, Alice!
```

Конкатенация в print() может быть удобнее, чем перечисление аргументов через запятую, так как при использовании запятой в print() Python добавляет пробелы по умолчанию. Конкатенация строк позволяет точнее контролировать, что будет выведено на экран.

Умножение строк

В Python строки можно умножать на числа. Эта операция повторяет строку указанное количество раз:

```
text = "ha"
laugh = text * 3
print(laugh)  # hahaha
```

Таким образом, умножение строки на число полезно, когда нужно создать строку, повторяющуюся несколько раз. Это может быть использовано для создания визуальных эффектов или шаблонов в текстовых выводах. Например:

```
print("-" * 10)  # ----------
```

Этот метод полезен для оформления вывода, автоматизации повторяющихся действий и других целей, где нужно повторить строку определенное количество раз.