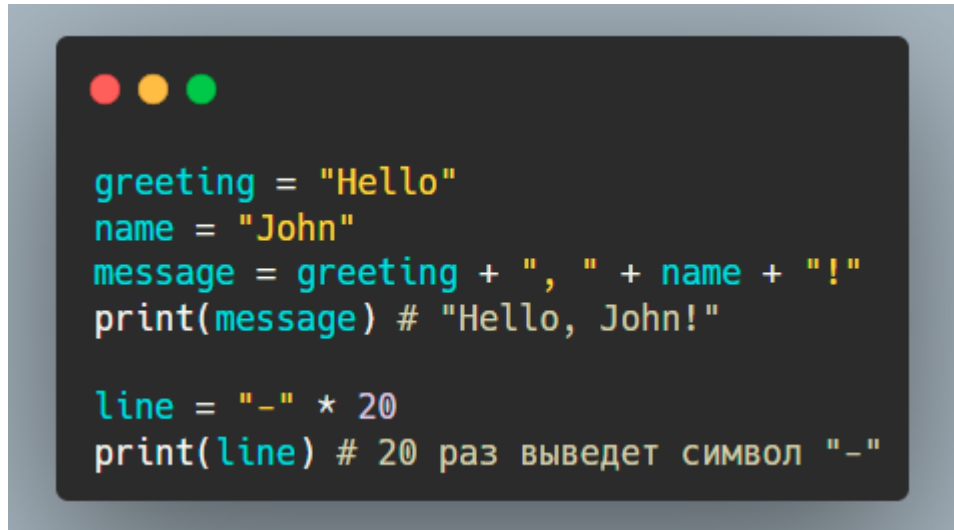


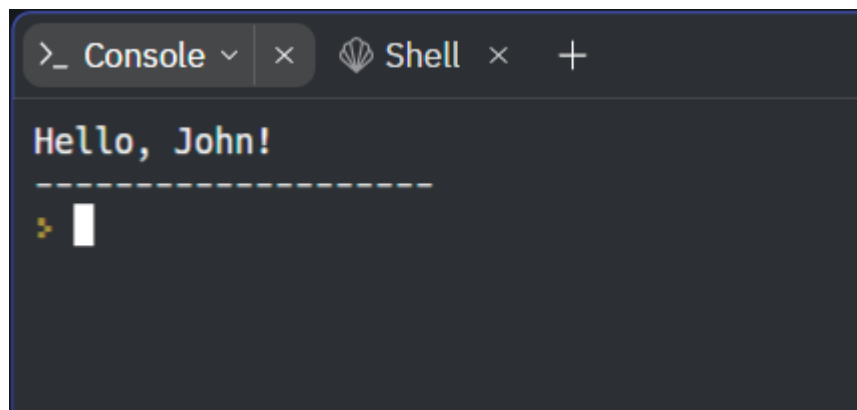
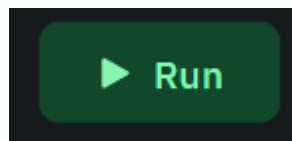
Строки в Python поддерживают множество операций и методов, которые позволяют работать с ними. Например, можно объединять строки с помощью оператора "+" или умножать строку на целое число, чтобы создать новую строку, содержащую несколько копий исходной строки.



```
greeting = "Hello"
name = "John"
message = greeting + ", " + name + "!"
print(message) # "Hello, John!"

line = "-" * 20
print(line) # 20 раз выведет символ "-"
```

Запустим, нажав на кнопку «Run».

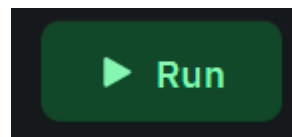


Строки также поддерживают индексацию, то есть можно получить доступ к отдельным символам строки по их индексу (начинающемуся с 0). Кроме того, можно использовать срезы (slice), чтобы получить подстроку из строки.

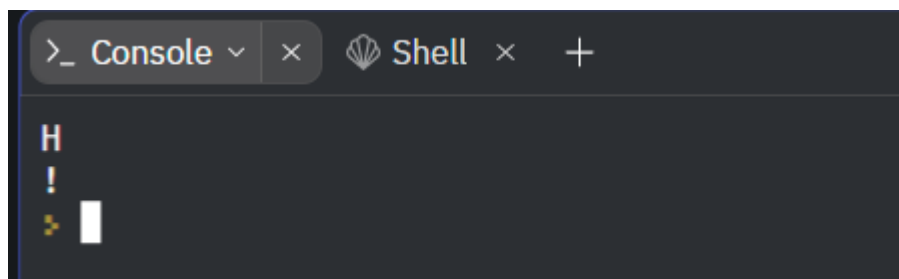
A screenshot of a Python code editor window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. The code inside is:

```
text = "Hello, world!"
print(text[0])
print(text[-1])
```

Запустим, нажав на кнопку «Run».



В результате получаем первый и последний символ строки

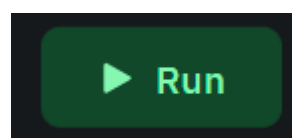


Строки в Python также имеют множество методов, которые позволяют выполнять различные операции с ними. Например, методы `upper()` и `lower()` позволяют преобразовывать строку в верхний или нижний регистр соответственно, метод `strip()` удаляет пробелы с начала и конца строки, метод `replace()` заменяет часть строки на другую строку и т.д.

A screenshot of a Python code editor window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. The code inside is:

```
text = "    Hello, world!    "
print(text.strip()) # Hello, World! (без пробелов)
print(text.replace("world", "John")) # Hello, John (с пробелами)
```

Запустим, нажав на кнопку «Run»



```
>_ Console × Shell × +  
Hello, World!  
    Hello, John!  
> █
```

Стоит заметить, что пробелы во второй строке остались.

Применение методов осуществляется через точку после строки или переменной.

```
text = "    Hello, world!    "  
print(text.strip()) # Hello, World! (без пробелов)  
print(text.replace("world", "John")) # Hello, John (с пробелами)
```

Выделяют следующие методы для работы со строками

Функция или метод	Назначение
<code>S = 'str'; S = "str"; S = '''str'''; S = """"str""""</code>	Литералы строк
<code>S = "s\np\ta\nbbb"</code>	Экранированные последовательности
<code>S = r"C:\temp\new"</code>	Неформатированные строки (подавляют экранирование)
<code>S = b"byte"</code>	Строка байтов
<code>S1 + S2</code>	Конкатенация (сложение строк)
<code>S1 * 3</code>	Повторение строки
<code>S[i]</code>	Обращение по индексу
<code>S[i:j:step]</code>	Извлечение среза
<code>len(S)</code>	Длина строки

S.find(str, [start],[end])	Поиск подстроки в строке. Возвращает номер первого вхождения или -1
S.rfind(str, [start],[end])	Поиск подстроки в строке. Возвращает номер последнего вхождения или -1
S.index(str, [start],[end])	Поиск подстроки в строке. Возвращает номер первого вхождения или вызывает ValueError
S.rindex(str, [start],[end])	Поиск подстроки в строке. Возвращает номер последнего вхождения или вызывает ValueError
S.replace(шаблон, замена[, maxcount])	Замена шаблона на замену. maxcount ограничивает количество замен
S.split(символ)	Разбиение строки по разделителю
S.isdigit()	Состоит ли строка из цифр
S.isalpha()	Состоит ли строка из букв
S.isalnum()	Состоит ли строка из цифр или букв
S.islower()	Состоит ли строка из символов в нижнем регистре
S.isupper()	Состоит ли строка из символов в верхнем регистре
S.isspace()	Состоит ли строка из неотображаемых символов (пробел, символ перевода строки ('\f'), "новая строка" ('\n'), "перевод каретки" ('\r'), "горизонтальная табуляция" ('\t') и "вертикальная табуляция" ('\v'))

S.istitle()	Начинаются ли слова в строке с заглавной буквы
S.upper()	Преобразование строки к верхнему регистру
S.lower()	Преобразование строки к нижнему регистру
S.startswith(str)	Начинается ли строка S с шаблона str
S.endswith(str)	Заканчивается ли строка S шаблоном str
S.join(список)	Сборка строки из списка с разделителем S
ord(символ)	Символ в его код ASCII
chr(число)	Код ASCII в символ
S.capitalize()	Переводит первый символ строки в верхний регистр, а все остальные в нижний
S.center(width, [fill])	Возвращает отцентрированную строку, по краям которой стоит символ fill (пробел по умолчанию)
S.count(str, [start],[end])	Возвращает количество непересекающихся вхождений подстроки в диапазоне [начало, конец] (0 и длина строки по умолчанию)
S.expandtabs([tabsize])	Возвращает копию строки, в которой все символы табуляции заменяются одним или несколькими пробелами, в зависимости от

	текущего столбца. Если TabSize не указан, размер табуляции полагается равным 8 пробелам
S.lstrip([chars])	Удаление пробельных символов в начале строки
S.rstrip([chars])	Удаление пробельных символов в конце строки
S.strip([chars])	Удаление пробельных символов в начале и в конце строки
S.partition(шаблон)	Возвращает кортеж, содержащий часть перед первым шаблоном, сам шаблон, и часть после шаблона. Если шаблон не найден, возвращается кортеж, содержащий саму строку, а затем две пустых строки
S.rpartition(sep)	Возвращает кортеж, содержащий часть перед последним шаблоном, сам шаблон, и часть после шаблона. Если шаблон не найден, возвращается кортеж, содержащий две пустых строки, а затем саму строку
S.swapcase()	Переводит символы нижнего регистра в верхний, а верхнего – в нижний
S.title()	Первую букву каждого слова переводит в верхний регистр, а все остальные в нижний

<code>S.zfill(width)</code>	Делает длину строки не меньшей <code>width</code> , по необходимости заполняя первые символы нулями
<code>S.ljust(width, fillchar=" ")</code>	Делает длину строки не меньшей <code>width</code> , по необходимости заполняя последние символы символом <code>fillchar</code>
<code>S.rjust(width, fillchar=" ")</code>	Делает длину строки не меньшей <code>width</code> , по необходимости заполняя первые символы символом <code>fillchar</code>
<code>S.format(*args, **kwargs)</code>	Форматирование строки

Циклы

В Python есть два основных вида циклов: цикл `for` и цикл `while`.

Цикл `for`

Цикл `for` используется для перебора элементов в итерируемом объекте, таком как список, кортеж или строка. Синтаксис цикла `for` в Python выглядит следующим образом:

```

for <переменная> in <итерируемый объект>:
    <тело цикла>

```

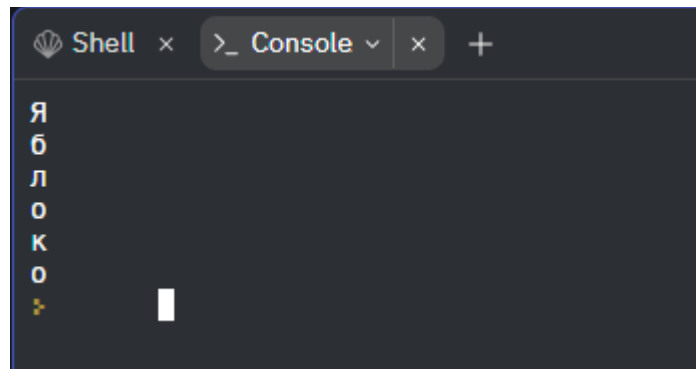
Например, следующий код выводит все элементы списка:

```

for char in "Яблоко":
    print(char)

```

В результате выполнения кода будут построчно выведены символы из строки «Яблоко».



Цикл while

Цикл while используется для выполнения блока кода до тех пор, пока указанное условие истинно. Синтаксис цикла while в Python выглядит следующим образом:

```
while <условие>:  
    <тело цикла>
```

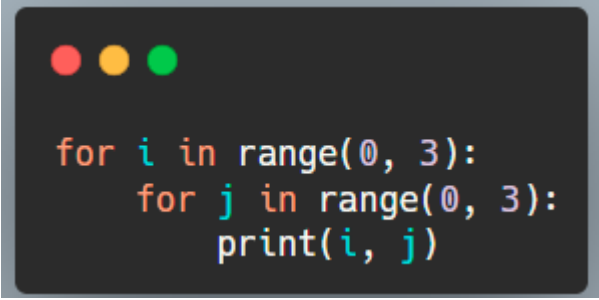
Например, следующий код выводит числа от 1 до 5 с помощью цикла while:

```
i = 1  
while i <= 5:  
    print(i)  
    i += 1
```

Вложенные циклы

В Python можно использовать вложенные циклы - циклы, которые находятся внутри других циклов. Вложенные циклы используются, когда требуется обработать данные, которые имеют многомерную структуру, например, двумерные массивы или матрицы.

Синтаксис вложенных циклов очень простой – в теле одного цикла можно разместить другой цикл:



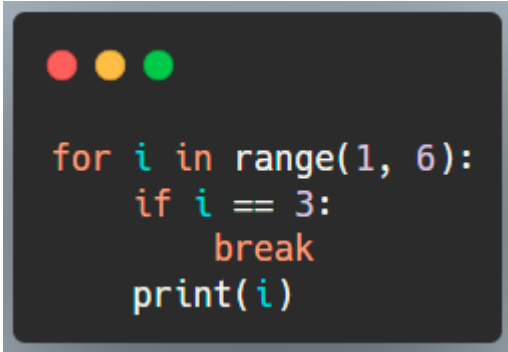
```
for i in range(0, 3):  
    for j in range(0, 3):  
        print(i, j)
```

В этом примере выводятся два столбца из 9 элементов, и для вывода всех его элементов используются вложенные циклы. Внешний цикл перебирает строки (i), а внутренний цикл - столбцы (j) массива.

Операторы break и continue

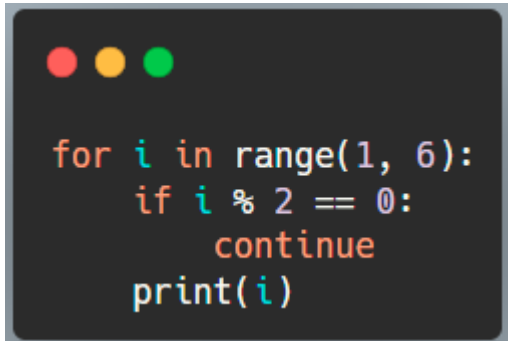
Внутри циклов можно использовать операторы break и continue.

Оператор break используется для прерывания выполнения цикла, если выполняется определенное условие. Например, следующий код выводит числа от 1 до 5, но прерывает цикл, если число 3:



```
for i in range(1, 6):  
    if i == 3:  
        break  
    print(i)
```

Оператор continue используется для пропуска текущей итерации цикла и перехода к следующей. Например, следующий код выводит только нечетные числа от 1 до 5:



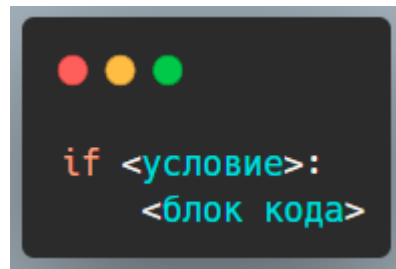
```
for i in range(1, 6):  
    if i % 2 == 0:  
        continue  
    print(i)
```

Условия

Условные конструкции в Python используются для выполнения различных действий в зависимости от выполнения определенных условий. В Python есть два основных оператора условия: if и else.

Оператор if

Оператор if позволяет выполнить определенный блок кода, если определенное условие истинно. Синтаксис оператора if выглядит следующим образом:

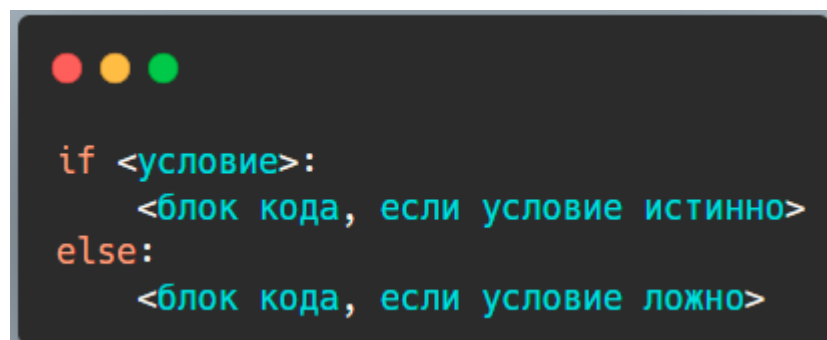
A code editor window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. It displays the Python if statement syntax:

```
if <условие>:  
    <блок кода>
```

```
if <условие>:  
    <блок кода>
```

Оператор if-else

Оператор if-else позволяет выполнить один блок кода, если условие истинно, и другой блок кода, если условие ложно. Синтаксис оператора if-else выглядит следующим образом:

A code editor window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. It displays the Python if-else statement syntax:

```
if <условие>:  
    <блок кода, если условие истинно>  
else:  
    <блок кода, если условие ложно>
```

```
if <условие>:  
    <блок кода, если условие истинно>  
else:  
    <блок кода, если условие ложно>
```

Оператор if-elif-else

Оператор if-elif-else позволяет проверить несколько условий и выполнить соответствующий блок кода для первого истинного условия. Синтаксис оператора if-elif-else выглядит следующим образом:

```
if <условие 1>:  
    <блок кода, если условие 1 истинно>  
elif <условие 2>:  
    <блок кода, если условие 2 истинно>  
else:  
    <блок кода, если все условия ложны>
```

Рассмотрим несколько задач для примера.

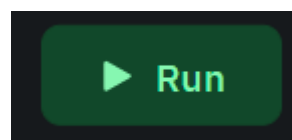
Задача 1.

Напишите программу, которая принимает строку и выводит ее длину.

```
s = input("Введите строку: ")  
print("Длина строки:", len(s))
```

В строку `s` считываем строчку с консоли и через `print()` выводим длину строки, передавая в функцию `len()` строку `s`.

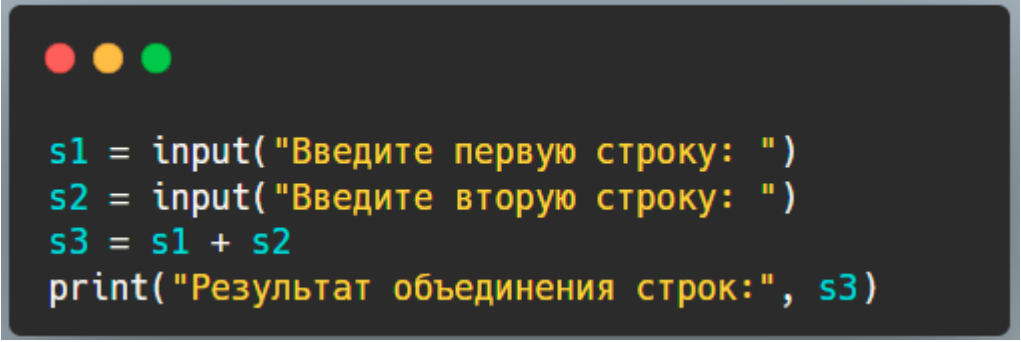
Запустим, нажав на кнопку «Run».



```
>_ Console v x Shell x +  
Введите строку: Строка  
Длина строки 6  
➤
```

Задача 2.

Напишите программу, которая принимает две строки, объединяет их и выводит результат.



```
s1 = input("Введите первую строку: ")
s2 = input("Введите вторую строку: ")
s3 = s1 + s2
print("Результат объединения строк:", s3)
```

Данный код запрашивает у пользователя две строки через консольный ввод с помощью функции `input()`.

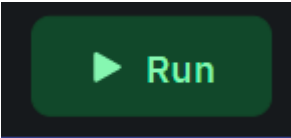
Затем строки сохраняются в переменные `s1` и `s2` соответственно.

Далее происходит операция объединения (конкатенации) строк с помощью оператора `+`. Результат сохраняется в переменной `s3`.

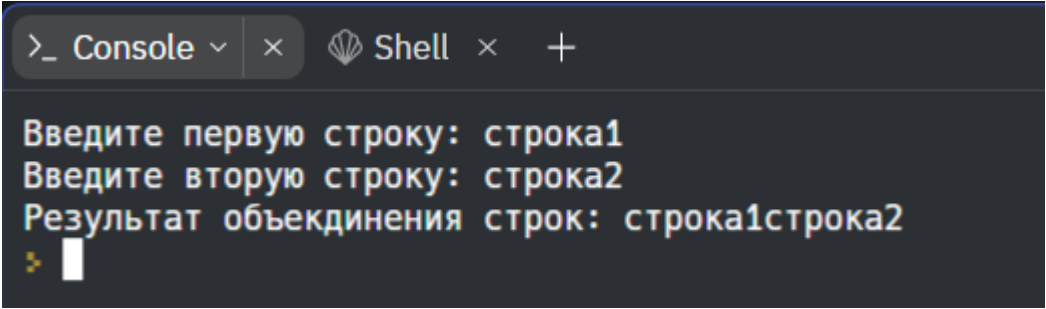
Наконец, результат объединения строк выводится на экран с помощью функции `print()`, в которой передается строка текста и значение переменной `s3`.

Таким образом, в результате выполнения кода на экране будет выведена строка, состоящая из двух строк, введенных пользователем.

Запустим, нажав на кнопку «Run».



Run



```
>_ Console × Shell × +
Введите первую строку: строка1
Введите вторую строку: строка2
Результат объекдинения строк: строка1строка2
>
```

Задачи для самостоятельного решения:

Задача 1.

Напишите программу, которая принимает две строки, и выводит их на экран, разделяя пробелом.

Задача 2.

Напишите программу на Python, которая запрашивает у пользователя число и выводит на экран таблицу умножения для этого числа. Если пользователь ввел отрицательное число или ноль, программа должна вывести сообщение об ошибке и завершиться.