Университет Сириус
Колледж

Основы программирования
Лекция 2.

Строки. Списки. Оператор ветвления.

Университет Сириус Колледж

Строки

- Неизменяемый итерируемый тип данных
- Могут быть объявлены разными способами
- Поддерживают синтаксис срезов (так же, как и списки)
- Поддерживают специальные символы
- Поддерживают интерполяцию
- Поддерживают форматирование
- Отдельные символы имеют кодировку по таблице ASCII или Unicode

Университет Сириус Колледж

Таблица ASCII

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.A	.B	.C	.D	.E	.F
0.	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1.	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2.		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3.	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4.	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5.	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6.	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7.	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Университет Сириус Колледж

Строки. Способы объявления

```
1  # Строки могут быть заключены как в одинарные, так и в двойные кавычки
2  message = 'hello'
3
4  recepient = "world"
```

Университет Сириус Колледж

Строки. Способы объявления

```
# Длинные строки можно заключить в любые кавычки, повторенные трижды
greeting = """
    - Hello
      there!
"""

answer = '''
    - General
      Kenobi!
 '''
```

Университет Сириус Колледж

Строки. Способы объявления

```
1  # Строки могут быть перенесены в коде с помощью символа переноса
2  same_dialogue = 'hello \
3    again'
```

Университет Сириус Колледж

Индексы и срезы (slices)

- Получение символа из строки имеет следующий синтаксис: `имя_переменной[индекс]`
- Срез имеет синтаксис: `имя_переменной[начало:конец:шаг]`

Университет Сириус Колледж

Индексы и срезы (slices)

- Получение символа из строки имеет следующий синтаксис: `имя_переменной[индекс]`
- Срез имеет синтаксис: `имя_переменной[начало:конец:шаг]`

При этом начало включено, конец - нет!

Университет Сириус Колледж

Индексы и срезы (slices). Примеры

Последовательность	a	b	c	d	e	f	g	Результат
Индексы	0(-7)	1(-6)	2(-5)	3(-4)	4(-3)	5(-2)	6(-1)	
`[:]`	+	+	+	+	+	+	+	abcdefg
`[::-1]`	+	+	+	+	+	+	+	gfedcba
`[::2]`	+		+		+		+	aceg
`[1::2]`		+		+		+		bdf
`[:1]`	+							a
`[-1:]`							+	g
`[3:4]`				+				d
`[-3:]`					+	+	+	efg
`[-3:1:-1]`			+	+	+			edc
`[2:5]`			+	+	+			cde

Университет Сириус Колледж

Специальные экранируемые символы в строках

Символ	Описание
\ в самом конце строки	Игнорируется, строка продолжается на новой строке
\\	Сам символ обратного слеша (остается один символ)
\'	Апостроф (остается один ‘)
\"	Кавычка (остается один символ ")
\n	Новая строка (перевод строки)
\r	Возврат каретки
\t	Горизонтальная табуляция

Если вы хотите, чтобы обратный слеш игнорировался, можете использовать **raw string** ("сырую" строку). Пример:

```
path = r'\path\to\folder\'
```

Университет Сириус Колледж

Методы строк

Метод	Описание	Пример
`str.upper()`	Преобразует строку в верхний регистр	`"hello".upper() → "HELLO"`
`str.lower()`	Преобразует строку в нижний регистр	`"HELLO".lower() → "hello"`
`str.capitalize()`	Первый символ в верхний регистр, остальные - нижний	`"hello".capitalize() → "Hello"`
`str.title()`	Каждое слово с заглавной буквы	`"hello world".title() → "Hello World"`
`str.strip()`	Удаляет пробелы с обоих концов	`" hello ".strip() → "hello"`
`str.lstrip()`	Удаляет пробелы слева	`" hello ".lstrip() → "hello "`
`str.rstrip()`	Удаляет пробелы справа	`" hello ".rstrip() → " hello"`
`str.replace(old, new)`	Заменяет подстроку	`"hello".replace("l", "x") → "hexxo"`

Университет Сириус Колледж

Методы строк

Метод	Описание	Пример
`str.split(sep)`	Разделяет строку по разделителю	`"a,b,c".split(",") → ["a", "b", "c"]`
`str.join(iterable)`	Объединяет элементы через строку	`",".join(["a", "b"]) → "a,b"`
`str.startswith(prefix)`	Проверяет начало строки	`"hello".startswith("he") → True`
`str.endswith(suffix)`	Проверяет конец строки	`"hello".endswith("lo") → True`
`str.find(sub)`	Ищет подстроку, возвращает индекс	`"hello".find("l") → 2`
`str.rfind(sub)`	Ищет подстроку с конца	`"hello".rfind("l") → 3`
`str.index(sub)`	Как find(), но вызывает ошибку	`"hello".index("l") → 2`
`str.count(sub)`	Считает количество вхождений	`"hello".count("l") → 2`
`str.isalpha()`	Только буквы?	`"hello".isalpha() → True`
`str.isdigit()`	Только цифры?	`"123".isdigit() → True`

Университет Сириус Колледж

Методы строк

Метод	Описание	Пример
`str.isalnum()`	Буквы или цифры?	`"abc123".isalnum() → True`
`str.islower()`	Все символы в нижнем регистре?	`"hello".islower() → True`
`str.isupper()`	Все символы в верхнем регистре?	`"HELLO".isupper() → True`
`str.isspace()`	Только пробельные символы?	`" ".isspace() → True`
`str.zfill(width)`	Дополняет нулями слева	`"42".zfill(5) → "00042"`
`str.center(width)`	Центрирует строку	`"hi".center(5) → " hi "`
`str.ljust(width)`	Выравнивает по левому краю	`"hi".ljust(5) → "hi "`
`str.rjust(width)`	Выравнивает по правому краю	`"hi".rjust(5) → " hi"`

Университет Сириус Колледж

Методы строк

Метод	Описание	Пример
`str.isalnum()`	Буквы или цифры?	`"abc123".isalnum() → True`
`str.islower()`	Все символы в нижнем регистре?	`"hello".islower() → True`
`str.isupper()`	Все символы в верхнем регистре?	`"HELLO".isupper() → True`
`str.isspace()`	Только пробельные символы?	`" ".isspace() → True`
`str.zfill(width)`	Дополняет нулями слева	`"42".zfill(5) → "00042"`
`str.center(width)`	Центрирует строку	`"hi".center(5) → " hi "`
`str.ljust(width)`	Выравнивает по левому краю	`"hi".ljust(5) → "hi "`
`str.rjust(width)`	Выравнивает по правому краю	`"hi".rjust(5) → " hi"`

Важно! *Так как строки являются неизменяемым типом все методы возвращают новую строку, а не изменяют существующую.*

Университет Сириус Колледж

Методы строк

Метод	Описание	Пример
`str.isalnum()`	Буквы или цифры?	`"abc123".isalnum() → True`
`str.islower()`	Все символы в нижнем регистре?	`"hello".islower() → True`
`str.isupper()`	Все символы в верхнем регистре?	`"HELLO".isupper() → True`
`str.isspace()`	Только пробельные символы?	`" ".isspace() → True`
`str.zfill(width)`	Дополняет нулями слева	`"42".zfill(5) → "00042"`
`str.center(width)`	Центрирует строку	`"hi".center(5) → " hi "`
`str.ljust(width)`	Выравнивает по левому краю	`"hi".ljust(5) → "hi "`
`str.rjust(width)`	Выравнивает по правому краю	`"hi".rjust(5) → " hi"`

Важно! *Так как строки являются неизменяемым типом все методы возвращают новую строку, а не изменяют существующую.*

Университет Сириус Колледж

Структуры данных

- Структуры данных описывают точку зрения пользователя на представление данных.
- Любая структура данных имеет ограниченный набор операций, которые на этой структуре можно выполнять.
- Структура данных имеет ряд условных правил (ограничений), определяющих соответствие данных этой структуре

Университет Сириус Колледж

Структуры данных в Python

В языке Python существует ряд основных структур данных:

- Списки (lists);
- Кортежи (tuples);
- Словари (dictionaries);
- Множества (sets).

Университет Сириус
Колледж

Список (list)


Arrays
DEVS.LOL
Python Lists
Linked lists
What the hell is this?

Университет Сириус Колледж

Список (list)

- Список служит для того, чтобы хранить объекты (данные) в определенном порядке, особенно если порядок или содержимое могут изменяться.
- Списки можно изменять: можно добавить или удалить элементы, а также перезаписать существующие.
- Примеры списков:

```
>> empty_list  = []
>> numbers = [1, 2, 3, 4, 5]
```

Также список можно объявить с помощью `list()` :

```
>> new_empty_list = list()
```

Университет Сириус Колледж

Инициализация списка

Создать список определенного размера и сразу заполнить его значениями можно с помощью оператора умножения (*)

```
>> zeros = [0] * 5
>> zeros
[0, 0, 0, 0, 0]
```

Университет Сириус Колледж

Инициализация списка

Создать список определенного размера и сразу заполнить его значениями можно с помощью оператора умножения (*)

```
>> ones = [1] * 5
>> ones
[1, 1, 1, 1, 1]
```

Университет Сириус Колледж

Инициализация списка

Создать список определенного размера и сразу заполнить его значениями можно с помощью оператора умножения (*)

```
>> repetition = [1, 2, 3] * 3
>> repetition
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

Университет Сириус Колледж

Создание списков

Список также можно получить из других итерируемых типов данных.

Например, из строки:

```
>> list('cat')
['c', 'a', 't']
```

Или из кортежа (tuple):

```
>> new_tuple = ('1', '2', '3')
>> list(new_tuple)
['1', '2', '3']
```

Университет Сириус Колледж

Создание списков

Также многие стандартные функции возвращают список.

Например, функция `split()` применительно к строке разбивает эту строку на сегменты, по указанному пользователем разделителю, или же по пробелу, если таковой не указан:

```
>> today = '25/07/2021'
>> today.split('/')
['25', '07', '2021']
```

Университет Сириус Колледж

Получение элементов списка

Получить элемент списка можно, указав его смещение:

```
>> numbers = ['0', '5', '10']
>> numbers[1]
'5'
```

Не забывайте, что индексация элементов начинается с нуля, а отрицательные индексы отсчитываются с конца строки:

```
>> numbers[-1]
'10'
```

Университет Сириус Колледж

Вложенные списки (nested lists)

В качестве значений списки могут содержат другие списки. Например, у нас есть список маленьких птиц, а есть список птиц побольше.

```
>> small_birds = ['hummingbird', 'sparrow']
>> bigger_birds = ['pigeon', 'crow']
```

Объявим список `all_birds` , содержащий всех наших птиц:

```
>> all_birds = [small_birds, bigger_birds]
>> all_birds
[['hummingbird', 'sparrow'], ['pigeon', 'crow']]
```

Университет Сириус Колледж

Изменение элемента списка

Элемент списка можно изменить, также обратившись к нему по его смещению:

```
>> words = ['hello', 'world']
>> words[1] = 'python'
>> words
['hello', 'python']
```

Университет Сириус Колледж

Извлечение элементов с помощью диапазона смещений

Из списка можно извлечь последовательность, использовав диапазон смещений (срез):

```
>> numbers = [0, 1, 2, 3, 4, 5]
>> numbers[0:2]
[0, 1]
>> numbers[::-1]
[5, 4, 3, 2, 1, 0]
```

Университет Сириус Колледж

Добавление элементов в список

Университет Сириус Колледж

Добавление элементов в список

- С помощью метода `append()`:

Университет Сириус Колледж

Добавление элементов в список

- С помощью метода `append()`:

```
>> numbers.append(6)
[0, 1, 2, 3, 4, 5, 6]
```

Университет Сириус Колледж

Добавление элементов в список

- С помощью метода `append()`:

```
>> numbers.append(6)
[0, 1, 2, 3, 4, 5, 6]
```

- С помощью оператора `+=`:

Университет Сириус Колледж

Добавление элементов в список

- С помощью метода `append()`:

```
>> numbers.append(6)
[0, 1, 2, 3, 4, 5, 6]
```

- С помощью оператора `+=`:

```
>> numbers += [7]
[0, 1, 2, 3, 4, 5, 6, 7]
```

Университет Сириус Колледж

Добавление элементов в список

- С помощью метода `append()`:

```
>> numbers.append(6)
[0, 1, 2, 3, 4, 5, 6]
```

- С помощью оператора `+=`:

```
>> numbers += [7]
[0, 1, 2, 3, 4, 5, 6, 7]
```

- С помощью метода `extend()` (добавление списка к существующему):

Университет Сириус Колледж

Добавление элементов в список

- С помощью метода `append()`:

```
>> numbers.append(6)
[0, 1, 2, 3, 4, 5, 6]
```

- С помощью оператора `+=`:

```
>> numbers += [7]
[0, 1, 2, 3, 4, 5, 6, 7]
```

- С помощью метода `extend()` (добавление списка к существующему):

```
>> numbers.extend([8,  9])
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Университет Сириус Колледж

Добавление элементов в список

- С помощью метода `append()`:

```
>> numbers.append(6)
[0, 1, 2, 3, 4, 5, 6]
```

- С помощью оператора `+=`:

```
>> numbers += [7]
[0, 1, 2, 3, 4, 5, 6, 7]
```

- С помощью метода `extend()` (добавление списка к существующему):

```
>> numbers.extend([8,  9])
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- С помощью метода `insert()`:

Университет Сириус Колледж

Добавление элементов в список

- С помощью метода `append()`:

```
>> numbers.append(6)
[0, 1, 2, 3, 4, 5, 6]
```

- С помощью оператора `+=`:

```
>> numbers += [7]
[0, 1, 2, 3, 4, 5, 6, 7]
```

- С помощью метода `extend()` (добавление списка к существующему):

```
>> numbers.extend([8,  9])
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- С помощью метода `insert()`:

```
>> numbers.insert(0,  -1)
[-1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Университет Сириус Колледж

Удаление элементов из списка

Университет Сириус Колледж

Удаление элементов из списка

- С помощью `del` (удаление по индексу):

Университет Сириус Колледж

Удаление элементов из списка

- С помощью `del` (удаление по индексу):

```
>> del numbers[0]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Университет Сириус Колледж

Удаление элементов из списка

- С помощью `del` (удаление по индексу):

```
>> del numbers[0]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- С помощью `remove()` (удаление по значению):

Университет Сириус Колледж

Удаление элементов из списка

- С помощью `del` (удаление по индексу):

```
>> del numbers[0]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- С помощью `remove()` (удаление по значению):

```
>> numbers.remove(8)
[0, 1, 2, 3, 4, 5, 6, 7, 9]
```

Университет Сириус Колледж

Удаление элементов из списка

- С помощью `del` (удаление по индексу):

```
>> del numbers[0]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- С помощью `remove()` (удаление по значению):

```
>> numbers.remove(8)
[0, 1, 2, 3, 4, 5, 6, 7, 9]
```

Удаляется только первое вхождение элемента. Если элемент не найден, генерирует исключение `ValueError`

Университет Сириус Колледж

Удаление элементов из списка

- С помощью `del` (удаление по индексу):

```
>> del numbers[0]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- С помощью `remove()` (удаление по значению):

```
>> numbers.remove(8)
[0, 1, 2, 3, 4, 5, 6, 7, 9]
```

Удаляется только первое вхождение элемента. Если элемент не найден, генерирует исключение `ValueError`

- С помощью `clear()`:

Университет Сириус Колледж

Удаление элементов из списка

- С помощью `del` (удаление по индексу):

```
>> del numbers[0]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- С помощью `remove()` (удаление по значению):

```
>> numbers.remove(8)
[0, 1, 2, 3, 4, 5, 6, 7, 9]
```

Удаляется только первое вхождение элемента. Если элемент не найден, генерирует исключение `ValueError`

- С помощью `clear()`:

```
>> numbers.clear()
[]
```

Университет Сириус Колледж

Удаление элементов из списка

- С помощью метода `pop([index])`, возвращающей элемент по указанному индексу и удаляющей его из списка:

```
>> numbers.pop(0)
0
>> numbers
[1, 2, 3, 4, 5, 6, 7, 9]
```

Параметр `index` по умолчанию равен `-1`, поэтому функция `pop()` возвращает последний элемент списка.

```
>> numbers.pop()
9
>> numbers
[1, 2, 3, 4, 5, 6, 7]
```

Университет Сириус Колледж

Поиск элемента в списке

С помощью метода `index()`:

```
>> numbers.index(7)
6
```

> *Если элемент не найден, то будет выведено исключение `ValueError`.*

С помощью метода `count()`:

```
>> numbers.count(3)
1
```

Университет Сириус Колледж

Сортировка списка

С помощью метода `sort([key, reverse])` или функции `sorted(list, [key, reverse])`.

Если параметры `key` и `reverse` не указаны, то сортируются элементы списка по неубыванию.

Параметр `reverse` указывает на то, что список должен быть отсортирован в обратном порядке.

```
>> numbers = [3, 1, 2, 4, 5]
>> numbers.sort()
>> numbers
[1, 2, 3, 4, 5]
```

Университет Сириус Колледж

Сортировка списка

С помощью метода `sort([key, reverse])` или функции `sorted(list, [key, reverse])`.

Если параметры `key` и `reverse` не указаны, то сортируются элементы списка по неубыванию.

Параметр `reverse` указывает на то, что список должен быть отсортирован в обратном порядке.

```
>> numbers = [3, 1, 2, 4, 5]
>> numbers.sort()
>> numbers
[1, 2, 3, 4, 5]
>> numbers.sort(reverse=True)
>> numbers
[5, 4, 3, 2, 1]
```

Университет Сириус Колледж

Сортировка списка

Параметр `key` указывает на то, что список должен быть отсортирован по функции `key`.

Функция `key` должна принимать один аргумент.

```
>> numbers = [3, 1, 2, 4, 5]
>> numbers.sort(key=lambda x: -x)
>> numbers
[5, 4, 3, 2, 1]
```

Университет Сириус Колледж

Сортировка списка

Параметр `key` указывает на то, что список должен быть отсортирован по функции `key`.

Функция `key` должна принимать один аргумент.

```
>> numbers = [-3, 1, -2, 4, -5]
>> numbers.sort(key=lambda x: x ** 2)
>> numbers
[1, -2, -3, 4, -5]
```

Университет Сириус Колледж

Сортировка списка

Параметр `key` указывает на то, что список должен быть отсортирован по функции `key`.

Функция `key` должна принимать один аргумент.

```
>> student_tuples = [
...     ('john', 'A', 15),
...     ('jane', 'B', 12),
...     ('dave', 'B', 10),
... ]
>> sorted(student_tuples, key=lambda student: student[2])
[('dave', 'B', 10), ('jane', 'B', 12), ('john', 'A', 15)]
```

Университет Сириус Колледж

Функции списков

- `len()` - длина списка

```
>> numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>> len(numbers)
10
```

- `max()` - максимальное значение

```
>> numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>> max(numbers)
9
```

Университет Сириус Колледж

Функции списков

- `min()` - минимальное значение

```
>> numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>> min(numbers)
0
```

- `sum()` - сумма значений

```
>> numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>> sum(numbers)
45
```

Университет Сириус Колледж

Списковое включение (List comprehension)

Списковое включение — это некий синтаксический сахар, позволяющий упростить генерацию последовательностей (списков, множеств, словарей, генераторов).

```
новый_список = [«операция» for «элемент списка» in «список»]
```

- операция подразумевает некие действия, которые вы собираетесь применить к каждому элементу списка;
- элемент списка — каждый отдельный объект списка;
- список — последовательность, элементы которой вы планируете подвергнуть операции (это не обязательно должен быть `list`, подойдет любой итерируемый объект).

```
>> old_prices = [120, 550, 410, 990]
>> discount = 0.15
>> new_prices = [int(product * (1 - discount)) for product in old_prices]
>> new_prices
[102, 467, 348, 841]
```

Университет Сириус Колледж

Списковое включение (List comprehension)

Списковое включение — это некий синтаксический сахар, позволяющий упростить генерацию последовательностей (списков, множеств, словарей, генераторов).

```
новый_список = [«операция» for «элемент списка» in «список»]
```

- операция подразумевает некие действия, которые вы собираетесь применить к каждому элементу списка;
- элемент списка — каждый отдельный объект списка;
- список — последовательность, элементы которой вы планируете подвергнуть операции (это не обязательно должен быть `list`, подойдет любой итерируемый объект).

```
>> old_prices = [120, 550, 410, 990]
>> discount = 0.15
>> new_prices = [int(product * (1 - discount)) for product in old_prices]
>> new_prices
[102, 467, 348, 841]
```

Университет Сириус Колледж

Списковое включение (List comprehension)

Списковое включение — это некий синтаксический сахар, позволяющий упростить генерацию последовательностей (списков, множеств, словарей, генераторов).

```
новый_список = [«операция» for «элемент списка» in «список»]
```

- операция подразумевает некие действия, которые вы собираетесь применить к каждому элементу списка;
- элемент списка — каждый отдельный объект списка;
- список — последовательность, элементы которой вы планируете подвергнуть операции (это не обязательно должен быть `list`, подойдет любой итерируемый объект).

```
>> old_prices = [120, 550, 410, 990]
>> discount = 0.15
>> new_prices = [int(product * (1 - discount)) for product in old_prices]
>> new_prices
[102, 467, 348, 841]
```

Университет Сириус Колледж

Списковое включение (List comprehension)

Списковое включение — это некий синтаксический сахар, позволяющий упростить генерацию последовательностей (списков, множеств, словарей, генераторов).

```
новый_список = [«операция» for «элемент списка» in «список»]
```

- операция подразумевает некие действия, которые вы собираетесь применить к каждому элементу списка;
- элемент списка — каждый отдельный объект списка;
- список — последовательность, элементы которой вы планируете подвергнуть операции (это не обязательно должен быть `list`, подойдет любой итерируемый объект).

```
>> old_prices = [120, 550, 410, 990]
>> discount = 0.15
>> new_prices = [int(product * (1 - discount)) for product in old_prices]
>> new_prices
[102, 467, 348, 841]
```

Университет Сириус Колледж

Списковое включение
Условие в конце включения

```
новый_список = [«операция» for «элемент списка» in «список» if «условие»]
```

Такой вариант использования условий позволяет отсечь часть элементов итератора. Новый список будет короче первоначального. По сути, к той же конструкции, которая приведена выше, добавляется условие if.

```
>> numbers = [121, 544, 111, 99, 77]
>> number11 = [num for num in numbers if num % 11 == 0]
>> number11
[121, 99, 77]
```

Следует обратить внимание, что условие может быть только одно (т. е. здесь невозможно использовать `elif`, `else` или другие `if`, как мы могли бы сделать в циклах).

Университет Сириус Колледж

Списковое включение
Условие в начале включения

Если требуется не фильтрация данных по какому-то критерию, а изменение типа операции над элементами последовательности, условия могут использоваться в начале генератора списков.

```
новый_список = [«операция» if «условие» for «элемент списка» in «список»]
```

```
>> from string import ascii_letters
>> letters = 'hытфтрцзqп'
>> is_eng = [
...     f'{letter}-ДА' if letter in ascii_letters else f'{letter}-НЕТ'
...     for letter in letters
... ]
>> print(is_eng)
['h-ДА', 'ы-НЕТ', 't-ДА', 'ф-НЕТ', 'т-НЕТ', 'r-ДА', 'ц-НЕТ', 'з-НЕТ', 'q-ДА',
```

Университет Сириус Колледж

Сложные списковые включения

```
>> words = ['Я', 'изучаю', 'Python']
>> letters = [letter for word in words for letter in word]
>> letters
['Я', 'и', 'з', 'у', 'ч', 'а', 'ю', 'P', 'y', 't', 'h', 'o', 'n']
```

```
>> table = [[x * y for x in range(1, 6)] for y in range(1, 6)]
>> table
[[1, 2, 3, 4, 5],
 [2, 4, 6, 8, 10],
 [3, 6, 9, 12, 15],
 [4, 8, 12, 16, 20],
 [5, 10, 15, 20, 25]]
```

Университет Сириус Колледж

Оператор ветвления

```
1  if number > 0:
2      print('Число больше нуля')
```

Университет Сириус Колледж

Оператор ветвления

```
1  if number > 0:
2      print('Число больше нуля')
3  else
4      print('Число меньше либо равно нулю')
```

Университет Сириус Колледж

Оператор ветвления

```
if number > 0:
    print('Число больше 0')
elif number == 0:
    print('Число равно 0')
else
    print('Число меньше 0')
```

Университет Сириус Колледж

Оператор ветвления

```
if number > 0:
    print('Число больше 0')
elif number == 0:
    print('Число равно 0')
else
    print('Число меньше 0')
```

В Python отсутствует конструкция `switch/case`. Однако начиная с Python 3.10 был введен оператор сопоставления с шаблонами (pattern matching statement) `match/case`.

Университет Сириус Колледж

Операторы сравнения

Оператор	Действие	Пример
>	**Больше чем**: `True`, если левый операнд больше правого	x > y
<	**Меньше чем**: `True`, если левый операнд меньше правого	x < y
==	**Равно**: `True`, если операнды равны между собой	x == y
!=	**Не равно**: `True`, если операнды не равны между собой	x ≠ y
>=	**Больше или равно**: `True`, если левый операнд больше или равен правому	x ⩾ y
<=	**Меньше или равно**: `True`, если левый операнд меньше или равен правому	x ⩽ y

Университет Сириус Колледж

Логические операторы

Оператор	Действие	Пример
and	**И**: `True`, если оба операнда `True`	`x and y`
or	**ИЛИ**: `True`, если хотя бы один из операндов `True`	`x or y`
not	**НЕ**: `True`, если операнд `False`	`not x`

Университет Сириус Колледж

Операторы вхождения и тождественности

Оператор	Действие	Пример
`in`	`True`, если значение или переменная есть в последовательности	`5 in x`
`not in`	`True`, если значения или переменной нет в последовательности	`5 not in x`
`is`	`True`, если операнды идентичны (указывают на один объект)	`x is true`
`is not`	`True`, если операнды не идентичны (не указывают на один объект)	`x is not true`

Университет Сириус Колледж

Ленивые операции

Логические операции `and` и `or` являются ленивыми, то есть вычисляют только столько операндов, сколько необходимо для получения результата.

```
>> a = 0
>> b = 5
>> a ≠ 0 and b / a > 5
False
```

В данном примере вторая часть выражения `b / a > 5` не вычисляется, так как первый операнд `a ≠ 0` уже дал результат `False`, и для операции `and` этого достаточно.