Открыть ящик ПИТОНА

Долженкова Мария Львовна,
кандидат технических наук, заведующий кафедрой ЭВМ
Нижегородова Маргарита Владимировна,
кандидат педагогических наук, доцент кафедры САУ
Шмакова Наталья Александровна,
старший преподаватель кафедры САУ

Поговорим о данных

На прошлой лекции мы вспомнили, что программы есть последовательность действий, которую необходимо выполнить над данными для получения результата решения задачи. Все данные, которыми оперируют программы, относятся к определенным типам.

Тип данных – фундаментальное понятие теории программирования.

> Тип данных определяет множество значений, набор операций, которые можно применять к таким значениям и способ реализации хранения значений и выполнения операций.

Если достаточно формально подходить к вопросу о типизации языка Python, то можно сказать, что он относится к языкам с неявной сильной динамической типизацией.

Неявная типизация означает, что при объявлении переменной вам не нужно указывать ее тип, интерпретатор сам определяет тип при инициализации переменной.

```
a = 1 # целое число
b = 1.2 # число вещественное (с дробной частью)
c = 'python' # строка
```

Сильная типизация не позволяет производить операции в выражениях с данными различных типов, нельзя складывать, например, строки и числа, нужно все приводить к одному типу.

При динамической типизации тип переменной определяется непосредственно при выполнении программы, то есть одна и та же переменная, при многократной инициализации, может связываться с объектами разных типов.

```
a = 10 # переменная a ссылается на целое число
a = 'hello' # переменная a теперь ссылается на строку
```

В Python типы данных можно разделить на встроенные и невстроенные, которые можно использовать при импортировании соответствующих модулей. К основным встроенным типам относятся:

1) Логические переменные (bool). Примитивный тип данных, который принимает два значения – истина (True) или ложь (False). True и False ведут себя как числа 1 и 0. Используются в качестве результата логических операций.

2) Числа.

– int – целое число. Это числа без дробной части, которые, говоря математическим языком, являются расширением натурального ряда, дополненного нул ем и отрицательными числами. Целые числа имеют неограниченную точность, могут принимать сколь угодно большие значения.

– float – число с плавающей точкой. Их еще называют вещественными числами. Вещественные числа придуманы для измерения непрерывных величин. Однако ни один из языков программирования не способен реализовать числа с бесконечной точностью, поэтому всегда есть место приближению с определенной точностью, из-за чего возможны такие ситуации: при выполнении print(0.3 + 0.3 + 0.3) на экране может быть получено значение 0.8999999999999999

– complex – комплексное число. Состоит из двух чисел с плавающей точкой, представляющих соответственно его действительную и мнимую части. Доступ к обеим частям комплексного объекта x обеспечивают атрибуты x.real и x.imag, доступные только для чтения.

```
z = complex(1, 2)
print(z) # вывод на экран: (1 + 2j)
```

3) Списки.

– list – список. Это структура данных для хранения объектов различных типов, напоминающая массив. Каждому объекту соответствует индекс (место) в списке. Длина списка может изменяться за счет добавления или извлечения объектов. Пустой список имеет нулевую длину. Первый элемент списка имеет индекс 0, последний – (n - 1), где n – длина списка.

```
list1 = [] # пустой список
list2 = [1, 'two', 3.2] # список из трех объектов: целого числа 1, строки 'two' и вещественного числа 3.2
list3 = [0] * 4 # список [0,0,0,0]
list4 = [1, 2, 3] * 2 # список [1, 2, 3, 1, 2, 3]
```

– tuple – кортеж. Представляет последовательность элементов, которая во многом похожа на список за тем исключением, что мы не можем добавлять, удалять и изменять элементы в кортеже.

```
tuple1 = (1, 'two', [1, 2]) # кортеж из трех объектов: целого числа 1, строки 'two' и списка из двух элементов.
s = 'today'
tuple2 = tuple(s) # кортеж ('t', 'o', 'd', 'a', 'y')
```

4) Строки – str. Это упорядоченная последовательность символов, которая предназначена для хранения информации в виде простого текста. К символам строки, как и к элементам списка, можно обращаться по индексу, первый символ имеет индекс 0.

```
s = '' # пустая строка
```

```
s = 'today' # строка их 5 символов s[0] – символ 't', s[3] – символ 'a'
```

5) Множества – set. Это набор неупорядоченных уникальных (неповторяющихся) объектов, объединенных по какому-либо признаку. Возможны два варианта задания множества.

```
set1 = {'1', '2', '3'} # множество из трех элементов
l = ['1', '2', '3'] # список из трех элементов
set2 = set(l) # множество из трех элементов
```

> При последовательном выполнении команд store = set('TikTok') и print(store) на экране увидим: {'o', 'T', 'i', 'k'}, – так как множество – это неупорядоченный набор, порядок символов каждый раз может меняться.

6) Словари – dict. Является набором данных, однако не считается последовательностью, так как представляет собой неупорядоченный набор пар ключ-значение. При этом каждый ключ в рамках одного словаря является уникальным. Объявим и инициализируем словарь из трех элементов и выведем на экран значение третьего элемента.

```
example_dict = {'keyOne': 'valueFirst', 'keyTwo': 'valueSecond', 'keyThree': 'valueThird'}
print(example_dict['keyThree'])
example_dict = dict(keyOne ='valueFirst', keyTwo = 'valueSecond', keyThree = 'valueThird')
print(example_dict['keyThree']) # на экране: 'valueThird'
```

Размер словаря может изменяться за счет добавления или извлечения элементов.

Неизменяемые vs изменяемые типы

В Python существуют изменяемые и неизменяемые типы.

> К неизменяемым (immutable) типам относятся: целые числа (int), числа с плавающей точкой (float), комплексные числа (complex), логические переменные (bool), кортежи (tuple), строки (str).
>
> К изменяемым (mutable) типам относятся: списки (list), множества (set), словари (dict).

Неизменяемые объекты обладают замечательным свойством: они не могут измениться в результате работы программы. Любая попытка изменить объект приведет к созданию копии.

Изменяемые объекты не обладают таким постоянством. Они, скорее, напоминают контейнер для хранения: контейнер оста ется на месте, а вот содержимое может сильно измениться.

Мы уже знаем, что при создании переменной, вначале создается объект определенного типа и с конкретным значением, который имеет уникальный идентификатор. Чтобы получить идентификатор объекта можно воспользоваться функцией id().

Программа	На экране будет выведено
a = 100 print(a) print('id ', id(a)) a = 150 print(a) print('id ', id(a)) a = a + 50 print(a) print('id ', id(a))	100 id 1672501744 150 id 16723011356 200 id 167230044529

То есть при выполнении операции присваивания, изменяется идентификатор, следовательно, создается новый объект, а ссылка на объект со значением 100 теряется, в этом состоит неизменяемость типа.

Еще один неизменяемый тип – строка. Если попробовать изменить символ в существующей строке, произойдет ошибка.

Программа	На экране будет выведено
s = 'Строка' print(s) print(s[0]) s = '1' + s print(s) print(s[0]) s[0] = '2'	Строка С 1Строка 1 Traceback (most recent call last): File "main.py", line 7, in <module> s[0]='2' TypeError: 'str' object does not support item assignment

Итак, неизменяемые типы не позволяют модифицировать содержимое ячейки памяти, на которую ссылается переменная. Для изменения значения создается новый объект. Память, занятая предыдущим значением, становится недоступной и освобождается сборщиком мусора.

Если тип данных изменяемый, то можно менять значение объекта. Например, создадим список [1, 2], а потом заменим второй элемент на 3.

Программа	На экране будет выведено
list1 = [1, 2] print(list1) print(id(list1)) print(id(list1[1])) list1[1] = 3 print(list1) print(id(list1)) print(id(list1[1]))	[1, 2] 139661321240128 139661324887360 [1, 3] 139661321240128 139661324887392

То есть объект, на который ссылается переменная list, изменен, а идентификатор сохранился. В качестве данных списка выступают не объекты, а ссылки на объекты, содержащие числа, – в первом случае 1 и 2, а во втором 1 и 3

(сравните идентификаторы list[1] в обоих случаях). Кроме того, изменяемый тип, может изменять размер – количество элементов (об этом позже). Рассмотренная ситуация проиллюстрирована рисунком 1.

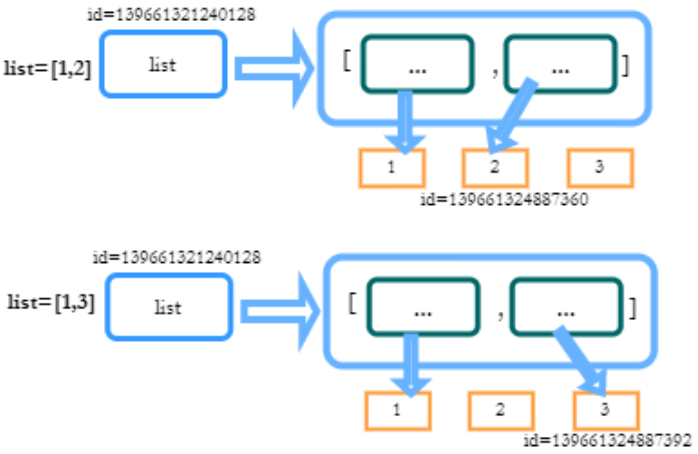

Рисунок 1 – Работа с элементами структуры

> ВАЖНО! Изменение мутируемого объекта не приводит к созданию копии.

Программа	На экране будет выведено
list1 = [1,2] list2 = list1 print(list1) print(id(list1)) print(list2) print(id(list2)) list1[1] = 3 print(list1) print(list2)	[1, 2] 139808273743360 [1, 2] 139808273743360 [1, 3] [1, 3]

Выполняем операции

Для начала введем некоторые определения.

> Операторы - специальные символы, которые выполняют арифметические и логические вычисления.
>
> Значения, на которые действует оператор, называются операндами

Арифметические операции

Арифметические операторы используются для выполнения математических операций – сложения, вычитания, умножения и так далее.

Оператор	Действие
+	Сложение двух операндов или унарный плюс
-	Вычитание правого оператора из левого или унарный минус
*	Умножение двух операндов

Оператор	Действие
/	Деление левого операнда на правый (результат всегда типа float)
%	Остаток от деления левого операнда на правый
//	Деление с округлением – деление, результат которого корректируется в меньшую сторону
**	Показатель степени – левый операнд возводится в значение правого операнда

В школьной математике мы изучали понятие «приоритет операции». Приоритет определяет то, в какой последовательности должны выполняться операции. Например, умножение и деление имеют больший приоритет, чем сложение и вычитание, а приоритет возведения в степень выше всех остальных арифметических операций: 2 * 2 ** 3 = 16.

Но нередко вычисления должны происходить в порядке, отличном от стандартного приоритета. В сложных ситуациях приоритет можно (и нужно) задавать круглыми скобками.

```
print(2 + 2) * 2 # => 8
print(3 ** (4 - 2)) # => 9
print(7 * 3 + (4 / 2) - (8 + (2 - 1))) # => 14.0
```

Главное при этом соблюдать парность, то есть закрывать скобки в правильном порядке. Это, кстати, часто становится причиной ошибок не только у новичков, но и у опытных программистов.

Приоритет операций:

1) скобки;
2) **(возведение в степень);
3) *, /, //, %;
4) +, -.

В Python множество составных операторов – это арифметические операции с присваиванием.

Оператор	Пример	Эквивалентно
=	x = 5	x = 5
+=	x += 5	x = x + 5
-=	x -= 5	x = x – 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5
%=	x %= 5	x = x % 5
//=	x //= 5	x = x // 5
**=	x **= 5	x = x ** 5

В языке Python имеются встроенные функции для работы с числами. Наиболее полезные, следующие.

Название	Описание
abs(x)	Вычисляет модуль числа x
round(x)	Округляет x до ближайшего целого
min(x1, x2,… ,x_n)	Находит минимальное, среди указанных чисел
max(x1, x2,… ,x_n)	Находит максимальное, среди указанных чисел
pow(x, y)	Возводит x в степень y

Инструкция	На экране будет выведено
print(abs(-2))	2
print(abs(-5*4))	20
print(min(3, 2, -5, 4))	-5
print(max(3, 2, abs(-5), 4))	5
print(pow(3, 2))	9
print(pow(9, 0.5))	3.0
print(round(3.45))	3
print(round(453.352, 1))	453.4
print(round(456, -1))	460
print(round(456, -2))	500

Python имеет довольно обширную библиотеку математических функций, представленную в модуле math, который можно импортировать в программу.

Особые операторы

В Python есть особые типы операторов: операторы тождественности и принадлежности.

is и is not – операторы тождественности в Python. Они проверяют, находятся ли два значения (или две переменные) по одному адресу в памяти (являются одним и тем же объектом).

> То, что две переменные равны, еще не значит, что они идентичны.

Оператор	Действие	Пример
is	Если операнды идентичны (указывают на один объект), возвращает значение True	x = 10 y = x print(x is y)
		True

Оператор	Действие	Пример
is not	Если операнды не идентичны (не указывают на один объект), возвращает значение True	x = (1, 2) y = (1, 2) print(x is not y)
		True

in и not in – операторы принадлежности в Python. Они проверяют, есть ли значение или переменная в последовательности (строке, списке, кортеже, множестве или словаре). Иначе говоря, проверяют вхождение элемента в набор данных.

Оператор	Действие	Пример
in	Если значение или переменная есть в последовательности возвращает значение True	x = [1, 2, 3, 4, 5] print(5 in x)
		True
not in	Если значения или переменной нет в последовательности возвращает значение True	x = [1, 2, 3, 4, 6] print(5 not in x)
		True

Давайте попробуем

Задача 1. В двух строках вводятся два числа (числа целые, положительные, не превышают 1000). Это катеты треугольника. Найдите гипотенузу этого треугольника.

Программа	На экране будет выведено
x,y = (input('Введите катеты через пробел \n').split()) print('%.2f' % (int(x) ** 2 + int(y) ** 2) ** 0.5)	Введите катеты через пробел 3 5 5.83

Задача 2. Напишите программу, которая считывает целое число и выводит текст: сначала фразу "The next number for the number ", затем введенное число, затем слово " is ", окруженное пробелами, затем следующее за введенным число, наконец, знак точки без пробела. Аналогично в следующей строке для предыдущего числа. Пробелы, знаки препинания, заглавные и строчные буквы важны!

Программа	На экране будет выведено
x = int(input('input number ')) print(f'The next number for the number {x} is {x+1}')	input number 66 The next number for the number

print(f'The prev number for the number {x} is {x-1}')	66 is 67 The prev number for the number 66 is 65

Задача 3. N школьников делят K яблок поровну, неделящийся остаток остается в корзинке. Сколько яблок останется в корзинке? Сколько яблок достанется каждому школьнику?

Программа	На экране будет выведено

N, k = input().split()
print('Каждому по', str(int(k) // int(N)), 'яблока' , sep = " ")
print('и', str(int(k) % int(N)), 'в корзине' , sep = " ")
	4 17
	Каждому по 4 яблока
	и 1 в корзине

Задача 4. Дано неотрицательное целое число. Найдите число десятков в его десятичной записи (то есть вторую справа цифру его десятичной записи).

Программа	На экране будет выведено
x = int(input('Исходное число ')) print((x // 10) % 10, ' десятков в нем ')	Исходное число 45687 8 десятков в нем

Задача 5. Электронные часы показывают время в формате h:mm:ss, то есть сначала записывается количество часов, потом обязательно двузначное количество минут, затем обязательно двузначное количество секунд. Количество минут и секунд при необходимости дополняются до двузначного числа нулями. С начала суток прошло n секунд. Выведите, что покажут часы.

Программа	На экране будет выведено

x = int(input('С начала суток прошло секунд '))
ans = str(x // 3600) + ':'
x = x % 3600
ans += '0' * abs((x // 60+50) // 60 -1) + str(x // 60) + ':'
x = x % 60
ans += '0' * abs((x + 50) // 60 - 1) + str(x)
print('Часы покажут ', ans)
	С начала суток прошло секунд 56789
	Часы покажут 15:46:29

Задача 6. Дано четырехзначное число. Определите, является ли его десятичная запись симметричной. Если число симметричное, то выведите 1, иначе выведите любое другое целое число.

Программа	На экране будет выведено
x = int(input('Введите число ')) ans = abs(x // 1000 - x % 10) + abs(x // 100 % 10 - x // 10 % 10) + 1 print(ans)	Введите число 1231 2

Что можно делать со строками?

Благодаря широкому распространению текста в повседневной жизни, строка является важным типом данных, присутствующим в мире программирования. Итак, в Python есть несколько способов задания строк.

Программа	На экране будет выведено
str1 = 'Hello1' print(str1) str2 = "Hello2" print(str2) str3 = '''Многострочные строки 1''' print(str3) str4 = """Многострочные строки2""" print(str4)	Hello1 Hello2 Многострочные строки 1 Многострочные строки2

Для работы со строками в Питоне предусмотрены операторы и специальные функции. Рассмотрим их.

Оператор «+» – это соединение двух строк или конкатенация. Конкатенация строк означает соединение строк вместе от первого до последнего символа для создания новой строки.

В Python довольно просто выполняется дублирование строки.

```
one = 'ай '
msg = one * 10
print(msg)
```

Для проверки наличия в строке той или иной подстроки, используется оператор «in». Он возвращает True, если подстрока присутствует и False, если отсутствует.

```
s = "abcdefg0123"
"abc" in s # True
'0' in s # True
'43' in s # False
```

Для сравнения строк используются операторы сравнения «==» (равно), «!=» (неравно), «<» (меньше), «>» (больше). При этом в строках рассматриваются по очереди отдельные символы и их регистр:

- цифра условно меньше, чем любая буква из алфавита;
- алфавитная буква в верхнем регистре меньше, чем буква в нижнем регистре;
- чем раньше буква в алфавите, тем она меньше (лексографический порядок).

Функция str(n) преобразует значение другого типа к строке (о преобразовании типов мы уже говорили). Преобразование числа в строку может быть полезным, когда мы, например, имеем дело с индексами или телефонными номерами.

Например, когда нам нужно объединить телефонный код страны и телефонный номер, но при этом мы не хотим их складывать

Для определения длины строки, то есть, числа символов в строке (включая управляющие символы), используется функция len(s).

При необходимости работать с кодами символов используют функции chr(s) для получения символа по его коду и ord(s) – для получение кода ASCII по символу. Зашифруем пароль для его дальнейшей передачи.

Программа	На экране будет выведено
pass1 = input('Введите пароль - кодовое слово из 5 символов ') pass1 = chr(ord(pass1[0]) + 3) + chr(ord(pass1[1]) + 4) + chr(ord(pass1[2]) + 5) + chr(ord(pass1[3]) + 4) + chr(ord(pass1[4]) + 3) print('Закодированное слово - ', pass1)	Введите пароль - кодовое слово из 5 символов слово закодированное слово - фпужс

Как мы уже говорили строка – упорядоченная последовательность и у каждого символа в строке есть свой порядковый номер – индекс, по которому мы можем его получить. Например, когда мы создаем строку

str1 = "Hello, Иван!"

то формируется следующая последовательность, изображенная на рисунке 2.

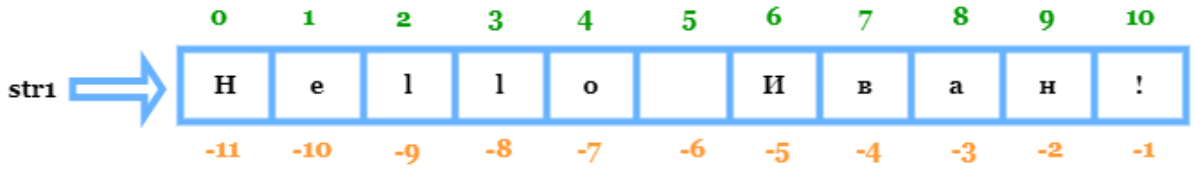

Рисунок 2 – Последовательность символов в строке

Например: str1[0] – это 'H', а str1[6] – это 'И'.

Но, если указать неверный индекс, например, str1[12], то возникнет ошибка. Поэтому здесь следует быть аккуратным и не выходить за пределы этого списка. В частности, последний «рабочий» индекс можно определить с помощью функции len – длины строки. То есть, к последнему индексу мы можем обратиться так: str1[len(str1) - 1].

Но это не очень удобно. Поэтому разработчики языка Python решили, что отрицательные индексы будут означать движение по строке с конца в начало. И предыдущую запись можно переписать так: str1 [-1]. Это намного удобнее. То есть, у строк есть и отрицательные индексы.

Работая со строками в программировании, регулярно приходится извлекать подстроки из строки. Подстрокой называется какая-то часть строки. Представим, что у нас есть дата в таком формате: 12-10-2022. Как извлечь из этой строки подстроку, в которую входит только год?

Если подумать логически, то для извлечения года, нам нужно посчитать индекс символа, с которого начинается год, и затем извлечь четыре символа. С этой целью используют срезы.

Срезы для строк в Python – это механизм, с помощью которого извлекается подстрока по указанным параметрам: начальный индекс (start), конечный индекс (stop) и шаг (step), которые указываются через символ «:». В нашем примере:

```
data = '12-10-2022'
print('год: ', data[6:10]) # 'год: 2022' или
print('год: ',data[-4:])
print('месяц: ',data[3:5]) # месяц: 10
print('день: ', data[:2]) # день: 12
```

Срезы очень – мощный механизм с большим количеством вариаций, например, если не указать вторую границу, то извлечение произойдет до конца строки, то же самое с первой границей (началом строки).

Шаг – третий необязательный параметр. По умолчанию он равен единице, но мы можем его изменить.

```
value = 'Hexlet'
value[1:5:2] # el
```

Шаг может быть отрицательным, в таком случае он берется с конца. Из этого вытекает самый популярный способ использования шага – переворот строки.

```
value[::-1] # 'telxeH'
```

Границы срезов можно указывать не только через числа, но и с использованием переменных.

```
value = 'Hexlet'
start = 1
end = 5
value[start:end] # 'exle'
```

С помощью срезов можно изменить строку msg = 'Hello World!' на строку 'Hello word!'. Это можно сделать так.

```
msg = msg[:6]+"w"+msg[7:]
```

В результате строка msg ссылается на новую измененную строку.

Теперь, когда у нас есть базовое понимание того, что представляют собой строки, давайте кратко разберем, что такое методы работы со строками.

> Вообще, метод – это встроенная функция, которая работает с объектом определенного типа данных. Существуют строковые методы, методы для работы с целыми числами, методы списков, словарей и так далее. Любой доступный метод можно вызвать следующим образом: объект.имя_метода(аргументы).

Например, мы хотим нашу строку msg = "Hello World!" преобразовать так, чтобы все символы отображались в верхнем регистре. Вызовем метод upper().

```
msg.upper()
```

Причем, сама переменная msg продолжает ссылаться на ту же самую неизмененную строку "Hello World!" – метод upper возвращает новую строку с заглавными буквами, не меняя прежней.

Если бы нам потребовалось изменить строку, на которую ссылается переменная, то мы бы могли использовать операцию присваивания: msg = msg.upper().

Основные методы для работы со строками представлены в таблице.

Метод	Описание	Пример
Работа с регистром		
capitalize()	Переводит первый символ строки в верхний регистр, а все остальные в нижний	print('привет, МИР!'.capitalize())
		Привет, мир!
lower()	Преобразование строки к нижнему регистру	print('Странная ФУНКция'.lower())
		странная функция
swapcase()	Переводит символы нижнего регистра в верхний, а верхнего – в нижний	print('Странная ФУНКция'.swapcase())
		сТРАННАЯ функЦИЯ
title()	Первую букву каждого слова переводит в верхний регистр, а все остальные в нижний	print('Странная ФУНКция'.title())
		Странная Функция
upper()	Преобразование строки к верхнему регистру	print('Странная ФУНКция'.upper())
		СТРАННАЯ ФУНКЦИЯ
Поиск и замена элементов		
find(sub[, start[, end]])	Возвращает индекс первого вхождения подстроки sub или -1 при отсутствии вхождения. Поиск идет в границах от start до end	text = "Wikipedia is a Python library that makes it easy to access and parse data from Wikipedia" print(text.find("Wikipedia"))
		0
rfind(sub[, start[, end]])	Возвращает индекс последнего вхождения подстроки sub или -1 при отсутствии вхождения. Поиск идет в границах от start до end	print(text.rfind("Wikipedia"))
		79

Метод	Описание	Пример
replace(sub, new)	Заменяет в текущей строке подстроку sub на новую подстроку new	print(text.replace("from Wikipedia", "from https://www.wikipedia.org/")
		Wikipedia is a Python library that makes it easy to access and parse data from https://www.wikipedia.org/
count(sub[, start[, end]])	Возвращает число вхождений подстроки sub в строку в заданном промежутке от start до end. При подсчете учитываются непересекающиеся вхождения	str2 = 'xxxx' print(str2. count('xx')) print(str2. count('xxx')) print(text.count('Wiki'))
		2 1 2
startswith(prefix[, start[, end]])	Возвращает True, если строка начинается на указанный префикс; в противном случае возвращает False	print(text. startswith('Wikipedia'))
		True
endswith(sufix[, start[, end]])	Возвращает True, если строка заканчивается на указанный суффикс; в противном случае возвращает False	print(text. endswith('Wiki'))
		False
Обрезка и выравнивание		
center(width, [fill])	Возвращает отцентрованную строку, по краям которой стоит символ fill (пробел по умолчанию)	print('xxx'.center(7, '/'))
		//xxx//
ljust(width, [fill]) rjust(width, [fill])	Делает длину строки не меньше width, по необходимости заполняя последние символы символом fill. Также есть метод rjust, выравнивающий по правому краю	print('xxx'.ljust(7, '*')) print('xxx'.rjust(7, '*'))
		xxx**** ****xxx
zfill(width)	Делает длину строки не меньше width, по необходимости заполняя первые символы нулями	print('123'.zfill(5))
		00123

Метод	Описание	Пример
strip([chars])	Удаление пробельных символов в начале и в конце строки, либо символов из параметра chars, если он передан. Также есть команды lstrip и rstrip, удаляющие символы только слева или только справа	print(' foo bo zooo '.strip()) print('(ни за что!:-))'.strip('()-: '))
		foo bo zooo ни за что
Проверка типа строки		
isalnum()	Состоит ли строка из букв и цифр	print('Wy7'.isalnum())
		True
isalpha()	Состоит ли строка из букв	print('Wy7'.isalpha())
		False
isdigit()	Состоит ли строка из цифр	print('12313'.isdigit())
		True
islower()	Состоит ли строка из символов в нижнем регистре	print('привет, мир'.islower())
		True
isspace()	Состоит ли строка из неотображаемых символов (пробел, табуляция, перенос строки и т.п.)	print('\n\t\r'.isspace())
		True
isupper()	Состоит ли строка из символов в верхнем регистре	print('ПРИВЕТ, МИР'.isupper())
		True
Нарезка и склейка строк		
split(sep, [maxsplit])	Разрезать строку по пробельным символам. Если указан sep, то по символам sep. Если указан maxsplit, то нарезается не более чем на указанное количество кусков. Также есть метод rsplit, который выдаст не более maxsplit кусков, считая справа	print(text.split(" "))
		['Wikipedia', 'is', 'a', 'Python', 'library', 'that', 'makes', 'it', 'easy', 'to', 'access', 'and', 'parse', 'data', 'from', 'Wikipedia']

Метод	Описание	Пример
'[char]'.join()	Склеивает все строки из переданного параметра, используя соединитель (возможно, пустой)	print("_".join(split_text))
		'Wikipedia_is_a_Python_library _that_makes_it_easy_to_access_and _parse_data_from_Wikipedia'
splitlines()	Нарезать большой кусок текста на строки по любому из символов переноса строки	print("""И дн ем, и ночью Кот уч еный...""".splitlines())
		['И дн ем, и ночью', 'Кот уч еный...']
partition()	Ищет шаблон в строке. И разрезает строку на три части: строки до шаблона, самого шаблона, и строки после	print('fooboozoo'.partition('boo'))
		('foo', 'boo', 'zoo')
Посимвольная замена		
maketrans(mask, shifr)	Созда ет таблицу замены для translate. Принимает на вход две строки одинаковой длины	trans = str.maketrans('бгл', 'bgl')
translate()	Используя изготовленную при помощи maketrans таблицу, произвести посимвольную замену	print('благодать'.translate(trans))
		blagодать

Давайте посмотрим

Задача 7. Вы тайный агент. Вам пришла шифровка. Необходимо расшифровать. Известно, что необходимо заменить в тексте все цифры 1 на слово 'раз', развернуть последовательность, заключенную между первым и последним вхождением буквы 'р' в обратном порядке.

```
text = input()
text = text.replace('1', 'раз')
n = text.find('р')
k = text.rfind('р')
text1 = text[n : k + 1]
text1 = text1[:: -1]
text = text[: n] + text1 + text[k + 1 :]
print(text)
```

Составляем список

Представьте, что вы собираетесь заехать за покупками в ближайший супермаркет. Что нужно сделать вначале? Составить список покупок. В Python есть такая структура данных.

Чтобы Python понял, что имеет дело со списком, нужно заключить все элементы в квадратные скобки ([]). Сами элементы разделяются запятыми.

Пусть нам необходимо составит список задач на сегодня. Мы можем создать пустой список tasks = [], либо записать в него первую задачу tasks = [‘прослушать лекцию’]. Во втором случае в списке есть один элемент (функция len(tasks) возвращает 1) и индекс этого элемента 0.

Список является изменяемым типом, поэтому допустимо следующее действие tasks[0] = 'посмотреть фильм'

Список в Python может содержать элементы любого типа. Он вполне может содержать в себе другие списки в качестве элементов. Это называется вложенностью, а подобные списки – вложенными. Пусть сейчас наш список должен включать список на сегодня и на завтра.

```
tasks = [['посмотреть фильм', 'сдать зачет'], ['навестить маму', 'купить книгу']]
```

Тогда каждый элемент будет иметь два индекса: номер во внешнем и номер во внутреннем списке. Например, tasks[0][1] – это 'сдать зачет'. А tasks[0] – все дела на сегодня: ['посмотреть фильм', 'сдать зачет'].

Кроме функции определения длины к спискам можно применить встроенные функции, такие как min(), max() и sorted().

– min() возвращает элемент, который шел бы первым, если бы список был упорядочен в алфавитном порядке.

– max() – который шел бы последним.

– sorted() упорядочивает элементы списка (если указать sorted(tasks[0], reverse = True), то будет создан новый список, отсортированный в обратном порядке.

Мы можем создавать новые списки, объединяя уже существующие. Добавим в список дел вчерашние задачи.

```
prev = ['вымыть окна', 'записаться на консультацию ']
tasks = [prev] + tasks
```

Можно узнать, есть ли среди задач на сегодня «сдать зачет».

```
'сдать зачет' in tasks[1]
```

Помимо уже названных встроенных функций, Python имеет еще и несколько специальных методов. Какие операции со списками мы проделываем чаще всего?

– Добавляем элементы (по одному или несколько сразу).

– Удаляем элементы.
– Меняем порядок элементов.

Мы можем добавлять элементы, вставляя их по одному в конец списка. Это делается при помощи метода append(). Давайте добавим задачу на сегодня в наш список.

```
tasks[1].append('сходить на тренировку')
```

Вместо того чтобы добавлять их по одному, мы можем воспользоваться методом extend() и добавить все элементы одного объекта в другой.

```
tasks.extend(['выполнить лабораторную', 'заправить машину'])
```

> Оператор «+» создает новый список, комбинируя списки, указанные в качестве операндов. А методы append() и extend() изменяют список, для которого они вызваны, и не возвращают новых списков.

Мы можем удалять элементы из списка по одному или даже группами. Метод pop() возвращает последний элемент списка и удаляет его, как показано ниже. После применения метода pop() последнего элемента в списке не стало.

```
last_element = tasks.pop()
```

Если бы мы хотели удалить элемент под определенным индексом, этот индекс можно было бы передать в метод pop() в качестве аргумента.

```
not_needed = tasks[1].pop(0)
```

Если нам не нужен доступ к значению удаляемого элемента, можно воспользоваться функцией del.

```
del tasks[0]
```

Предположим, мы знаем, какую именно задачу в списке мы передумали выполнять, но не знаем, какой у нее индекс. В подобных случаях можно воспользоваться методом remove() и удалить элемент по значению.

```
tasks[0].remove('сдать зачет')
```

Для удаления всех элементов из списка можно использовать tasks.clear(). Мы можем отсортировать, вызвав метод sort(). Вызов метода sort() изменяет существующий список и не создает нового.

Основные методы представлены в таблице.

Метод	Описание
append()	Добавляет элемент в конец списка
insert()	Вставляет элемент в указанное место списка
remove()	Удаляет элемент по значению
pop()	Удаляет последний элемент, либо элемент с указанным индексом

Метод	Описание
clear()	Очищает список (удаляет все элементы)
copy()	Возвращает копию списка
count()	Возвращает число элементов с указанным значением
index()	Возвращает индекс первого найденного элемента
reverse()	Меняет порядок следования элементов на обратный
sort()	Сортирует элементы списка

В списках, как и в строках можно применять срезы. Например, вы собираетесь в путешествие по Золотому кольцу России. Вам известен маршрут (список городов, которые планируется посетить, рисунок 3).

lst = ['Сергиев Посад', 'Владимир', 'Суздаль', 'Ярославль', 'Кострома', 'Иваново', 'Ростов Великий', 'Переславль-Залесский']

Рисунок 3 – Список городов

В каждом городе мы проведем по одному дню. Вот такая конструкция: lst[1:3] возвратит список из двух городов, в которых мы побываем во второй и третий день, ['Владимир', 'Суздаль'].

А вот так узнаем, где проведем предпоследний день тура: lst[-2:-1], получим ['Ростов Великий'].

Так как список – это изменяемый объект, то мы можем срезам присваивать новые значения. Допустим, туроператор решил изменить маршрут 3 и 4 дней.

Рисунок 4 – Новый маршрут

lst[2:5] = 'Углич', 'Муром'

В результате, получаем новый тур: ['Сергиев Посад', 'Владимир', 'Углич', 'Муром', 'Кострома', 'Иваново', 'Ростов Великий', 'Переславль-Залесский'].

И еще одна особенность. Помните?

lst1 = lst

Создает еще одну ссылку на наш объект список. Значит, любое изменение в списке lst приведет к модификации lst1. Чтобы избежать такого эффекта, необходимо создать копию списка и это можно сделать через срез.

lst1 = lst[:]

Словари

Словарь – это неупорядоченная последовательность произвольных объектов с доступом по ключу.

Его еще называют ассоциативным списком. Выше мы рассматривали список городов маршрута нашего путешествия. Каждый элемент его ассоциировался с индексом 0, 1, 2 и так далее. Это ассоциация, создаваемая по умолчанию. Пусть необходимо связать каждый город из маршрута с телефонным кодом. Например: Сергиев Посад – 49654, Владимир – 4922, Углич – 48532, Муром – 49234, Кострома – 4942, Иваново – 4932, Ростов Великий – 48536, Переславль-Залесский – 48535. Для таких ассоциаций удобно использовать словарь.

dict1 = {'Сергиев Посад': 49654, 'Владимир': 4922, 'Углич': 48532, 'Муром': 49234, 'Кострома': 4942, 'Иваново': 4932, 'Ростов Великий': 48536, 'Переславль-Залесский': 48535}

Можно создать словарь при помощи слияния двух списков, причем лишние значения будут проигнорированы.

Программа	На экране будет выведено
City = ['Москва', 'Киров'] Kod = [496, 8332, 922] dict2 = dict(zip(City, Kod)) print(dict2)	{'Москва': 496, 'Киров': 8332}

Таким образом, город – это ключ, а код – ассоциированное с ним значение. Создавая словарь, необходимо помнить о некоторых ограничениях.

– Данные, представляющие собой ключ словаря, должны быть уникальны внутри множества ключей этого словаря. Не должно быть двух одинаковых ключей.

– Ключ должен быть объектом неизменяемого типа, то есть строкой, числом или кортежем.

На значения нет никаких ограничений. Они не обязаны быть ни уникальными, ни неизменяемыми. Зная ключ, мы всегда можем определить соответствующее ему значение.

Например, чтобы узнать код Владимира достаточно обратиться к словарю.

print(Dict1[‘Владимир’]) # Результат 4922

Если попытаться ввести не существующий ключ, например, print(Dict1[‘Архангельск]) – получим ошибку исполнения программы.

Проверка вхождения ключа в словарь осуществляется операцией «in»: print('Москва' in Dict1) выведет False, так как ключ в словаре отсутствует, а print('Кострома' in Dict1) – True, так как город есть в словаре.

Для расширения словаря достаточно добавить в него пару ключ-значение. Ключ добавится в словарь, если его там не было или изменится его значение, если ключ в словаре уже присутствует.

Dict1['Москва'] = 496.

Чтобы удалить ключ из словаря достаточно написать del Dict1['Москва']. Удаление не существующего ключа приводит к ошибке, перед удалением лучше проверить его наличие.

Если необходимо упорядочить словарь по ключу – используется функция sorted().

Программа	**На экране будет выведено**
dict1 = {'Сергиев Посад': 49654, 'Владимир': 4922, 'Углич': 48532, 'Муром': 49234, 'Кострома': 4942, 'Иваново': 4932} print(sorted(dict1))	['Владимир', 'Иваново', 'Кострома', 'Муром', 'Сергиев Посад', 'Углич']

Основные методы для работы со словарями представлены ниже.

Метод	**Описание**	**Пример**
clear()	Удаляет все элементы из словаря	Dict1.clear() print(Dict1)
		{}
copy()	Возвращает копию словаря	dict3 = dict2.copy() print(dict3) print(dict2)
		{'Москва': 496, 'Киров': 8332} {'Москва': 496, 'Киров': 8332}
get(key [,d])	Возвращает значение ключа key. Если key не существует, возвращает d (по умолчанию None)	print(dict2.get('Москва', ооо)) print(dict2.get('Хабаровск', 0)
		496 0
items()	Возвращает новый объект элементов словаря в формате (ключ, значение)	print(dict2.items())
		dict_items([('Москва', 496), ('Киров', 8332)])
keys()	Возвращает новый объект с ключами словаря	print(dict2.keys())
		dict_keys(['Москва', 'Киров'])
pop(key [,d])	Удаляет элемент с ключом key и возвращает его значение или d,	s = dict2.pop('Москва', None) print(dict2) print(s)

Метод	Описание	Пример
	если key не найден. Если d не было обозначено и key не найден, вызывает ошибку KeyError	{'Киров': 8332} 496
popitem()	Удаляет и возвращает последнюю пару (ключ, значение). Вызывает ошибку KeyError, если словарь пустой	s = dict2.popitem() print(dict2) print(s)
		{ } ('Киров', 8332)
setdefault(key [,d])	Если ключ key есть в словаре, возвращает соответствующее ему значение. Если нет, добавляет в словарь элемент с ключом key и значением d и возвращает d (по умолчанию None)	dict2.setdefault('Хабаровск', 4212) print(dict2)
		{'Хабаровск': 4212}
update([other])	Обновляет словарь имеющимися парами ключ-значение из другого словаря, перезаписывая существующие ключи	dict2.update(dict3) print(dict2)
		{'Москва': 496, 'Киров': 8332, 'Хабаровск': 421}
values()	Возвращает новый объект со значениями словаря	print(dict2.values())
		dict_values([496, 8332, 4212])

Забегая вперед

Очень часто необходимо перебирать значения, хранящиеся в словарях. Для этого используется следующая конструкция.

```
for <переменная-ключ> in <словарь>:
    <действие>
```

Программа	На экране будет выведено
for city in dict1: print(city)	Сергиев Посад Владимир Углич Муром Кострома Иваново

В составе Python есть множество полезных модулей и библиотек, функции из которых можно использовать в своей программе. Мы уже говорили о модуле math, содержащем математические методы. Сейчас нам может потребоваться функция

генерации случайных значений. Набор таких функций реализован в модуле random. Например, если выполнить import random, то можно использовать функцию randint(start, end), генерирующую целое число в заданном диапазоне, или функцию randrange(n), генерирующую случайное целое число со значением от 0 до n - 1.

Давайте попробуем

Задача 8. Составим свою мини-Википедию. Представим, что у нас есть список знаменитостей, о каждом из которых имеется информация, например, год и страна рождения, страна проживания, сфера деятельности, самое большое достижение и так далее, причем не у всех набор информации одинаков. Напишите программу, которая по запросу пользователя из известного списка будет выводить имеющуюся о человеке информацию.

Программа	На экране будет выведено
contact = {'Россум': {'fullname': 'Гвидо ван Россум', 'birthday': '1956', 'country': 'Нидерланды', 'hipe': 'Создатель Python'}, 'Маск': {'fullname': 'Илон Рив Маск', 'birthday': '1971', 'country': 'ЮАР', 'success': '$209 млрд '}, 'Лукас': {'fullname': 'Джордж Уолтон Лукас ', 'birthday': '1944', 'film': 'Звездные войны', 'business': 'Кинорежисс ер'}} print("О ком вы хотите узнать?") print("\n".join(list(contact.keys()))) person = input() for one in contact[person]: print(one, contact[person][one])	О ком вы хотите узнать? Россум Маск Лукас Маск fullname Илон Рив Маск birthday 1971 country ЮАР success $209 млрд

Задача 9. Вам поручили написать часть модуля RPG игры с простой системой прокачки персонажа. На старте персонаж обладает минимальным списком сопротивлений. По ходу игры ему случайным образом выпадают заклинания, позволяющие пополнить их список.

Попробуем использовать несколько уже известных нам структур. Во-первых, список магических заклинаний ограничен и неизменяем, для его хранения буем использовать кортеж. Во-вторых, каждое заклинание связно с получением нового резиста, значит для установки соответствия можно использовать словарь. И, третье, все что имеет и получает персонаж, будем хранить в списке.

Программа	На экране будет выведено
import random have = ['паралич', 'безмолвие'] resist={'fire': 'огонь', 'ice': 'лед', 'death': 'смерть'} mag = tuple(resist.keys()) step = random.randrange(len(mag)) print('Выпало заклинание ', mag[step]) print('Бафаем новый резист ', resist[mag[step]]) have.append(resist[mag[step]]) print('Обновили арсенал:\n', have)	Выпало заклинание ice Бафаем новый резист лед Обновили арсенал: ['паралич', 'безмолвие', 'лед']

Практическая работа

1) Напишите программу, которая принимает на вход строку, преобразует ее в список и определяет, сколько слов в списке содержат заданную пользователем букву, причем только один раз.

2) Разработайте программу, которая использует информацию о ряде государств в Европе и Азии: название, столица, часть света, численность населения, площадь территории. Программа должна определять количество государств, расположенных в заданной части света, общую площадь государств в заданной части света, государство с минимальной численностью населения.

Контрольные вопросы

1) Что определяет тип переменной?

2) Python является языком с неявной сильной динамической типизацией. Что это означает?

3) Перечислите встроенные типы данных в Python.

4) Что такое множество? Какие операции существуют над множествами в Python?

5) Что такое кортеж? Какие операции над кортежами существуют в Python?

6) Что такое словарь? Какие операции над словарями существуют в Python?

7) Что такое список в Python? Опишите процесс создания списка.

8) Что такое изменяемые и неизменяемые объекты?

9) Какие существуют операции над строками в языке Python?

10) Определите приоритет операций и найдите значение выражения:

7 * 3**2 + 4 / 2 - (8 + 2 *4)/2