Оглавление

[Первые приложения. Базовый синтаксис и “Hello, World!” 2](#_Toc177999702)

[Управление потом исполнения команд (ветвление): введение 8](#_Toc177999703)

[Функции 21](#_Toc177999704)

[Списки 30](#_Toc177999705)

[Циклы 48](#_Toc177999706)

[Строки 57](#_Toc177999707)

[Словари 70](#_Toc177999708)

[Файлы 82](#_Toc177999709)

[Итоговое задание 91](#_Toc177999710)

Первые приложения. Базовый синтаксис и “Hello, World!”

Язык программирования python считается одним из наиболее простых в изучении языков программирования. Его использование позволяет использовать как функциональный, процедурный, так и объектно-ориентированные подходы к программированию. В языке поддерживаются высокоуровневые типы данных наряду с возможностью реализацией объектного подхода, что делает язык мощным инструментом для создания различных кроссплатформенных приложений.

Для того, чтобы начать создавать первые приложения на языке python, изучим базовые конструкции синтаксиса, необходимые для написания любых приложений.

Перед началом обучение введем условные обозначения:

>> - таким символом обозначается код, вводимый в IDE (Integrated development environment, интегрированная среда разработки, например, Spyder, Jupiter, Visual studio и тд).

ПРИВЕТ! - белые буквы на черном фоне обозначают результат вывода в консоль (специальная утилита, обеспечивающая поддержку прямой связи пользователя с ОС)

В рамках данного урока будут рассмотрены такие конструкции, как:

* Наиболее часто используемые типы данных (строки и числа)
* Функция print (вывод в консоль)
* Переменные
* Операции числами
* Комментарии
* Виды ошибок

В рамках данных работ будут написаны первые приложения на изучаемом языке Python.

**Наиболее часто используемые типы данных**

**Строки**

Строка – это текстовое сообщение, которое возможно обрабатывать с использованием встроенных инструментов языков программирования. В python строки могут быть заданы несколькими способами:

* Они могут быть заключены в одинарные кавычки ('…')
* Они могут быть заключены в двойные кавычки (“…”)

Оба способа абсолютно идентичны и можно использовать любой из них. Однако помните, что в начале и конце вводимой строки должны быть одинаковые кавычки!

Строкам соответствует тип данных String.

**Числа**

Числа в python задаются стандартно, как и во всех других языках программирования. Числа бывают двух типов: целые и дробные (или числа с плавающей запятой). Тип данных, соответствующий целым числам – integer (int). Данный тип содержит как отрицательные, так и положительные целые числа (например, -1, -2, 2, 3) и 0. При записи такого числа не допустимо использовать мантиссу (дробную часть). Такие числа используются при подсчете предметов, нумерации строк и т.д.

Числа с плавающей запятой, напротив, существуют для записи дробных чисел (чисел с плавающей запятой). Для этих целей наиболее часто в python используется тип данных float. Для задач, требующих большей точности измерений и вычислений используется тип данных double.

Числа с плавающей запятой используются для измерения относительных величин, статистических показателей и т.д. Например для расчета среднего роста учеников класса.

**Функция print**

Для общения с компьютером необходим инструмент, который бы выводил сообщения о своем текущем состоянии или операциях, которые он выполняет. С этой целью в python встроена функция print. Данная функция выводит строковые сообщения. Для задания строки, которую необходимо вывести, в скобках пишется текст сообщения в форме строки python (в кавычках):

>>print(“Прикоснись к прекрасному, обними программиста!”)

В данном фрагменте кода необходимо вывести в консоль выдержку из книги. Результатом исполнения данного кода будет следующее сообщение:

Прикоснись к прекрасному, обними программиста!

**Переменные**

Выше было рассмотрено, как можно ввести конкретное значение в python среду и вывести в консоль. Такие значения называются **литералами**. Например,

Прикоснись к прекрасному, обними программиста!

2

В данном случае были просто заданы строка и число, однако, они нигде не сохранены. Для возможности повторного использования этих значений необходимо выделить место в памяти компьютера, где будут хранится эти данные для дальнейшего использования. Для обращения к этим значениям выделяется некая область памяти компьютера, которой должно быть присвоено имя.

Для этих целей в языках программирования существуют переменные. Для задания переменных необходимо ввести имя переменной, знак равенства (=) и значение, которое туда необходимо поместить.

Например:

>>message\_string = "Приветик"

>># выведет "Приветик"

>>print(message\_string)

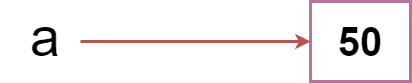
В данном случае в переменной message\_string сохранена строка, которая выводится в консоль посредством обращения к переменной.

На самом деле, В Python переменные – это символическое имя, которое является ссылкой или указателем на объект (область памяти).

Итак, если есть запись:

>>a=50

Это означает, что в определенном адресе ячейки записано значение 50, и мы можем получить доступ к данному числу с использованием имени a, то есть a ссылается на значение 50:



Рассмотрим другой случай. Например,

>>a=50

>>b=a

В этом случае, и a, и b ссылаются на одну и ту же область память (на объект – число 50). То есть:

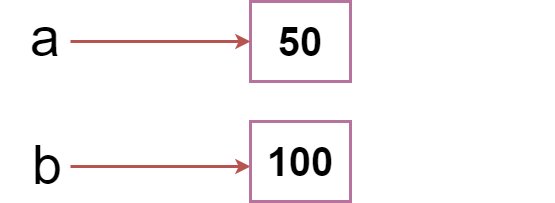


Но, если мы запишем так:

>>a=50

>>b=50

В этом случае, a и b будут ссылаться на разные объекты.



**Названия переменных**

Название переменных не может содержать пробелов или символов кроме подчеркивания (\_). Имена переменных не могут начинаться с чисел, но у них могут быть числа после первого символа (например, cool\_variable\_5 приемлемо). Имя переменной не может содержать специальные символы (!, @, #,%, ^, &, \*). Имена переменных не должны совпадать с зарезервированными словами python, например, не допустимо называть переменные print, type, def и т.д.. Так же необходимо запомнить, что python чувствителен к регистру, то есть My\_var и my\_var – это разные переменные!

При назначении имени переменной необходимо подбирать имя, отражающее ее назначение. Например, для переменной, содержащей возраст, подойдет имя age. Python поддерживает кодировку utf-8, в связи с чем имена могут содержать кириллические символы и это не вызовет ошибок. Однако, по стандартам чистого кода, кириллицу в названиях объектов в программе не рекомендуется.

При присвоении имени переменной, содержащей несколько слов, допустимо использовать несколько общепринятых подходов:

* Camel Case – каждое слово или сокращение в середине начинается с заглавной буквы, пробелов нет. Например – nameOfStudent, valueOfVaraible и т. д.
* Pascal Case – это то же самое, что и Camel Case, но здесь первое слово заглавное. Например – NameOfStudent и т. д.
* Snake Case – слова разделяются подчеркиванием. Например – name\_of\_student и т. д.

Значения в переменных можно модифицировать в процессе выполнения программы, что объясняет происхождение названия данной конструкции языка программирования.

>># Приветствие

>>message\_string = "Приветик"

>>print(message\_string)

>># Обновляем значение переменной

>>message\_string = "Hasta la vista"

>>print(message\_string)

В данном примере в начале в переменную message\_string введен текст приветствия, который затем выведен в консоль. После этого значение переменной заменено на прощание, а затем также выведено в консоль.

**Задание**

1. Создадим приложения, с которого начинается изучение всех классических языков программирования. На первом этапе выведите в консоль приветствие Hello, world!
2. Создайте переменную greeting и задайте ей значение Hello, world!
3. Выведите значение переменной greeting. Поздравляем! Вы создали свое первое приложение в python

**Форматирование строк при выводе в консоль**

Иногда, необходимо вывести какую-то строку одновременно со значением переменной. Например, есть переменная:

>>name = ‘Андрей’

Мы хотим вывести строку в консоль

Привет, Андрей!

В данном случае значение ‘Андрей’ содержится в переменной name, которую мы хотим вывести внутри строки. Для решения данной проблемы возможно 2 подхода:

1. Сложение строк (конкатенация)

Строки, как и числа, возможно складывать. Необходимо иметь в виду, что строку можно сложить только со строкой! Для сложения строк используется оператор +.

>> name = ‘Андрей’

>>print(‘Привет, ‘+name+’!’)

Данная команда позволит нам вывести желаемый результат.

1. Форматирование строк.

Для вывода переменной в строке можно использовать ключевое слово format. Для вывода желаемой нами строки будет использована следующая команда:

>> name = ‘Андрей’

>>print(‘Привет, {}!‘.format(name))

**Операции числами**

В рамках языка python можно выполнять арифметические действия над числами такие, как сложение, вычитание, деление и умножение (+, -, /, \*).

>>print(573 - 74 + 1)

>>print(25 \* 2)

>>print(10 / 5)

При выполнении последней операции в качестве результата будет получено число с плавающей запятой, так как деление преобразует целый тип данных integer в тип данных float (числа с плавающей запятой).

В результате деления на ноль будет выдана ошибка ZeroDivisionError.

**Комментарии**

Одним из наиболее полезных инструментов python являются комментарии. Они позволяют сделать код более читабельным и понятным. Для создания комментария необходимо ввести знак #. Все, что будет введено в строке после знака #, будет распознано, как комментарий, и компилятор будет игнорировать данные в этой строке при исполнении программы.

Комментарии могут:

• Обеспечить контекст, позволяющий понять, почему программа написана именно таким образом (например, для чего вводится дополнительная переменная и т.д.):

>># Эта переменная содержит информацию по количеству просмотров курса

>>viewers\_count = 0

• Помочь другим программистам прочитать Ваш код:

>># Данная переменная содержит информацию о сотрудниках

>>Personal\_count = 10

• Игнорировать часть кода при отладке приложения, чтобы найти, в какой строке происходит ошибка

>># value\_for\_test = 0

>>New\_value\_for\_test = 1

**Ошибки**

В языках программирования существует возможность нахождения ошибок кода, а также их разъяснения.

Python называет эти ошибки «Errors» и указывает на местоположение ошибки символом ^.

Существует 2 вида возможных ошибок в python:

* SyntaxError – означает, что что-то не так в тексте самой программы (синтаксисе) — например, несоответствующая пунктуация, которая не должна быть при данной операции. Также недостающая круглая скобка может привести к такого рода ошибке.
* NameError - происходит, когда интерпретатор Python не может распознать слово. Например переменную, которой не существует.

**Задание**

1. Необходимо расчитать, сколько плитки понадобится для проведения ремонта в комнате! Создать переменные length и width и присвоить в них значения 8 и 10 соответственно.
2. Оказалось, что произошла ошибка в расчетах и в длину необходимо 20 плиток. Какое количество плитки понадобится в этом случае?

Самостоятельно! Рассчитать площадь прямоугольника со сторонами 23, 13.

Управление потом исполнения команд (ветвление): введение

Каждый день мы сталкиваемся с проблемой выбора в зависимости от условий. Например, каким образом поехать на работу? Если общественным транспортом, то проблема с тем, что мы едем с большим количеством людей, зато будем на месте по расписанию. Если ехать на собственном автомобиле, то сама поездка будет гораздо комфортнее, но можем столкнуться с проблемой поиска парковки. Такие проблемы решаются с использованием правил «Если, то». Например, если на улице холодно, то нужно надеть куртку и т.д. Иными словами, некоторый порядок наших действий зависит от того, выполняется ли какое-нибудь условие.

В python последовательность действий называется набор команд, которые следуют в определенном порядке. Последовательность исполнения команд называется поток. До настоящего времени мы сталкивались только с прямым исполнением команд, то есть построчно сверху вниз. Но иногда нам необходимо некоторое ветвление потока исполнения, то есть выполнение некоторых команд только в том случае, если выполнено какое-то условие. Для этих целей используется составной оператор if/else.

В рамках данной темы вы научитесь управлять потоком исполнения команд с использованием логических выражений и оператора if/else.

**Операторы сравнения (логические операторы)**

Разберемся, что такое логические выражения. Под логическим выражением мы понимаем высказывание, которое задает какое-то условие. Например, выражение «На улице идет дождь». Данное выражение может быть либо правдой, либо ложью. В зависимости от того, есть ли дождь, принимается решение о необходимости зонтика. То есть логическое выражение может быть либо истинным, либо ложным.

Теперь, стало понятно, что такое логические выражения, необходимо научиться создавать их на Python. Мы можем создать логическое выражение, используя операторы сравнения.

Из названия видно, что данные операторы сравнивают 2 значения и возвращают либо True (если истина), либо False (если ложно).

Существует несколько видов операторов отношения. Для начала рассмотрим простой случай:

• Равно: == (обратите внимание, что равно состоит из двух знаков ==)

• Не равно:! =

Эти операторы сравнивают два элемента и возвращают True или False, если они равны или нет.

Мы можем создавать логические выражения, сравнивая два значения с помощью этих операторов:

>>>print (1 == 1)

True

>>>print(2 != 4)

True

>>> print(3 == 5)

False

>>> print('7' == 7)

False

Выше приведены примеры логических выражений. Как видно из примеров, результат исполнения команды, содержащей логическое выражение, является True или False.

**Обратите внимание!** True и False – это отдельный логический тип данных. До сих пор мы рассматривали только тип строка (string), тип целые числа (int) и тип числа с плавающей запятой (float). Логический тип называется **bool** и принимает только 2 вида значений: True или False. Обратите внимания, что данные значения пишутся с **заглавной** буквы. Напомним, что python чувствителен к регистру и True ≠true. Можно заметить, что при вводе их в редакторе кода (с заглавными буквами T и F), они отображаются цветом, отличным от переменных или строк.

**Задание:**

Определите, являются ли следующие логические выражения истинными или ложными.

Утверждение первое:

(6 \* 6) - 1 == 8 + 1

Утверждение второе:

13 - 7 != (3 \* 2) + 1

Утверждение третье:

3 \* (2 - 1) == 4 - 1

**Логические операторы больше и меньше.**

Выше мы рассмотрели 2 типа операторов сравнения: равно и не равно. Кроме них существуют и другие операторы сравнения:

• Больше чем:>

• Меньше чем: <

• Больше или равно:> =

• Меньше или равно: <=

**Задание:**

Определите, являются ли следующие логические выражения истинными или ложными.

(6 \* 6) - 1 >= 8 + 1

Утверждение второе:

13 - 7 <= (3 \* 2) + 1

Утверждение третье:

3 \* (2 - 1) > 4 - 1

**Логические переменные**

Так как True и False являются отдельным особым типом данных bool, то соответственно, как и в случае с числами и строками, можно создавать переменные, содержащие значения истины или лжи.

True и False - единственные типы bool, и любая переменная, которой присвоено одно из этих значений, называется логической переменной. Логические переменные можно создавать несколькими способами. Самый простой способ - просто присвоить переменной True или False:

>>true\_condition = True

>>false\_condition = False

Можно установить переменную, равную логическому выражению.

>>bool\_condition\_1 = 5 != 7

>>bool\_condition\_2 = 1 + 1 != 2

>>bool\_condition\_3 = 3 \* 3 == 9

Эти переменные теперь содержат логические значения, поэтому при ссылке на них они будут возвращать только значения True или False того выражения, которое им было присвоено.

>>>print (bool\_three)

True

>>> print(bool\_four)

False

>>>print(bool\_five)

True

**Задание:**

1. Создайте переменную с именем bool\_variable и установите для нее значение true. Попробуйте вывести ее в консоль. Какая ошибка у вас вывелась? Почему?

2. Замените значение в bool\_variable на ‘true’ (в кавычках) Проверьте тип bool\_variable, используя функцию проверки типов type (bool\_variable).

Выведите значение в консоль. Какой тип теперь у переменной bool\_variable?

3. Почему это не логическая переменная!

Создайте переменную с именем bool\_variable\_2 и сделайте так, чтобы она имелв логический тип истины.

4. Проверьте тип bool\_variable\_2 и убедитесь, что вы успешно создали логическую переменную.

**Выражения if.**

Понимание логических переменных и выражений важно, потому что они являются строительными блоками при составлении проверочных утверждений с использованием условных операторов.

Вернемся к примеру с дождем.

Если пойдет дождь, то вы возьмете зонт

Если перевести на английский язык, то получим следующее утверждение:

If it rains, then you take an umbrella

Тогда «пойдет дождь» - это логическое выражение, и этот условный оператор проверяет, истинно ли оно.

Если "идет дождь" == Истина, то остальная часть условного оператора будет выполнена, и вы принесете зонтик.

На самом деле, это форма условного оператора:

If [it rains] then [take an umbrella]

Условный оператор if проверяет истинность выражения, следующим за ним. Язык программирования Python хорош тем, что очень близок к конструкциям естесственного английского языка.

В Python проверка условия будет выглядит очень похоже на пример с зонтиком, сформулированном на английском языке:

>>if is\_raining:

>> bring\_umbrella()

Обратите внимание, что вместо слова «то» после условия используется двоеточие «:». Это интерпретируется компьютером так, что дальше будет тот набор действий (команд), которые должны быть выполнены в случае истинности условия.

Обратите внимание! Действия, которые выполняются в случае истинности условия, должны быть выделены отступами, причем, их должно быть одинаковое количество. То есть сам if мы пишем от начала строки, а все, что выполняется в случае истинности, имеют одинаковый отступ от края строки. Например:

>>age = 13

>>if age >=18:

>> print(‘Добро пожаловать на сайт алкогольной продукции’)

>>print(‘проверка выполнена’)

В данном примере мы видим, что задается переменная age, далее проверяется условие, age больше либо равен 18. Если это истина, то тогда вывести в консоль строку «Добро пожаловать на сайт алкогольной продукции». Однако, переменная age равна 13, условие в if ложно, соответственно, строга не будет выведена в консоль, однако будет выведена строка «проверка выполнена», так как она является частью основного кода и не входит в оператор if. Это связано с тем, что последний print написан от начала строки и не имеет отступов, то есть будет выполняться в любом случае вне зависимости от проверки условия в операторе if. Результатом исполнения данного кода будет:

проверка выполнена

Давайте посмотрим на другой условный оператор:

>>if 10 == 5 \* 2:

>> print("выражение это значение истинное имеет")

Будет ли этот код выводить выражение это значение истинное имеет на терминал? Да, потому что условие оператора if 10 == 5 \* 2 истинно.

**Задание:**

Я работаю в компании, обслуживающей проблемы информационной безопасности на других предприятиях. К нам обратился директор маленькой рекламной компании. Основная проблема в том, что у них есть охранник Дмитрий, который устанавливает компьютерные игры на АРМ (автоматизированные рабочие места) сотрудников, когда тех нет на работе по долгу (находятся в отпуске) и играет всеми ночами напролет. Поэтому вас просят разработать приложение, которое проверяло введенные учетные и разграничивало права сотрудников. Для охранника Дмитрия просят сделать специализированное уведомление: «Дмитрий, твое рабочее место находится в другой комнате. Отойди от чужого компьютера и займись работой!».

1. Введем переменную «user\_name»
2. Далее введем переменную, которая выводит текст для Дмитрия. Назовем ее Dmitriy\_check
3. Введем переменную, которая хранит сообщение для других сотрудников, вошедших в систему: «Добро пожаловать»
4. Напишем оператор if, которые проверяет значение переменной user\_name
5. Вывести в консоль результат выполнения программы для user\_name= «Дмитрий» и для user\_name= «Ангелина».

Наша компания продолжает разрабатывать приложение по безопасности для рекламной компании. Теперь, если пользователь 3 раза ввел пароль неправильно, необходимо заблокировать систему. Для этого выполним следующие действия:

1. Введем переменную, которая фиксирует количество попыток ввода и назовем ее enter\_number
2. С использованием оператора if напишем программу, которая если enter\_number меньше 3, то пишите «Попробуйте еще раз. У вас осталось (3- enter\_number) попыток». Если количество попыток больше либо равно 3, то выводим «Вы превысили максимальное число попыток. Ваша учетная запись заблокирована. Для разблокировки обратитесь в службу поддержки».
3. Проверить работу программы с использованием вывода в консоль.

**Логические операторы: and**

Мы рассмотрели операторы сравнения. Они являются разновидностью логических операторов. Существуют и другие типы логических операторов.

Существуют ситуации, при которых необходимо проверять сразу несколько условий. Например, мы берем зонтик только в том случае, если собираемся выходить на улицу в дождь. То есть одновременно должно быть 2 истинных условия:

1. На улице идет дождь
2. Мне нужно выйти на улицу

Для построения более сложных условия используются логические операторы:

* and
* or
* not

Начнем с and.

and проверяет выполнение двух логических выражений одновременно (логическое и) и примет значение True, если оба его логических выражения имеют значение True, и значение False в противном случае.

Рассмотрим пример

Курица – это птица и панда – это животное.

Это логическое выражение состоит из двух меньших выражений: курица - птица, а панда - животное, оба из которых имеют значение True и связаны логическим оператором and (и), поэтому все выражение истинно.

Давайте посмотрим на пример некоторых оператора and в Python:

>>> (1 + 1 == 2) and (2 + 2 == 4)

True

>>> (1 + 1 == 2) and (2 < 1)

False

>>> (1 > 9) and (5 != 6)

False

>>> (0 == 10) and (1 + 1 == 1)

False

Обратите внимание, что во втором и третьем примерах, даже если часть выражения имеет значение True, все выражение в целом имеет значение False, потому что другой оператор имеет значение False. Четвертый оператор также имеет значение False, поскольку оба компонента имеют значение False.

**Задание:**

1. Проверить истинность следующих выражений:

(2 + 2 + 2 >= 6) and (-1 \* -1 < 0)

(4 \* 2 <= 8) and (7 - 1 == 6)

Результат проверки поместить в переменные statement\_one и statement\_two.

1. Вернемся к задаче с проверкой безопасности в рекламном агентстве. Усложним задачу. Теперь нужно проверять не только имя пользователя, но и номер АРМ.

Для этого введем номера рабочих мест 4х сотрудников этой компании:

Дмитрий номер АРМ 1

Ангелина номер АРМ 2

Василий номер АРМ 3

Екатерина номер АРМ 4.

1. Введем переменную ARM, обозначающую номер АРМ.
2. Создадим условный оператор if, при котором проверяем соответствие номера АРМ и имени пользователя:

Если номер АРМ и имя пользователя соответствуют, то вывести в консоль «Добро пожаловать!»

Если номер АРМ не совпадает, а имя пользователя не Дмитрий, то «Логин или пароль не верный, попробуйте еще раз»

Если Если номер АРМ не совпадает, а имя пользователя Дмитрий, то Дмитрий, твое рабочее место находится в другой комнате. Отойди от чужого компьютера и займись работой!».

**Логические операторы: or**

Логический оператор or (логическое или) подразумевает, что выполняется хотя бы одно условие, то есть имеет значение True, если любой из компонентов имеет значение True.

Рассмотрим выражение:

Курица – это птица или панда – это рыба.

Это утверждение состоит из двух выражений: курица - птица, которое является истинным, а панда - рыба, которое является ложным. Поскольку два выражения связаны оператором or, весь оператор является истинным. Только один компонент должен быть истинным, чтобы оператор или был истинным.

Если оператор or имеет два компонента True, он также является True.

Давайте взглянем на пару примеров на Python:

>>> True or (3 + 4 == 7)

True

>>> (1 - 1 == 0) or False

True

>>> (2 < 0) or True

True

>>> (3 == 8) or (3 > 4)

False

Обратите внимание, что каждый оператор or, имеющий хотя бы один компонент True, имеет значение True, но последний оператор имеет два компонента False, поэтому он равен False.

**Задание:**

1. Проверить истинность следующих выражений:

(2 - 1 > 3) or (-5 \* 2 == -10)

(9 + 5 <= 15) or (7 != 4 + 3)

**Выражение else**

Как вы можете видеть из своей работы с задачей по информационной безопасности, как только вы начинаете включать в функцию множество операторов if, код становится немного загроможденным и неуклюжим. К счастью, есть и другие инструменты, которые мы можем использовать для управления потоком исполнения.

Операторы else позволяют описать, что необходимо сделать в случае, когда условия **не** выполняются.

Операторы else всегда появляются вместе с операторами if. Рассмотрим наш пример пробуждения, чтобы увидеть, как это работает:

>>if weekday:

>> wake\_up("6:30")

>>else:

>> sleep\_in()

Таким образом, мы можем создавать операторы if, которые позволяют выбирать, какие команды будут исполняться при условии истинности условий, а какие при условии ложности. Это избавляет нас от необходимости писать операторы if для каждого возможного условия, вместо этого мы можем написать общий оператор else для всех случаев, когда условие не выполняется.

Вернемся к примеру с сайтом алкогольной продукции. Давайте доработаем код так, чтобы в случае, если возраст меньше 18, выводил сообщение «Приносим наши извинения, мы не продаем продукцию лицам моложе 18 лет». Тогда наш код будет выглядеть следующим образом:

>>age = 13

>>if age >=18:

>> print(‘Добро пожаловать на сайт алкогольной продукции’)

>>else:

>> print(‘Приносим наши извинения, мы не продаем продукцию лицам моложе 18 лет’)

Обратите внимания на отступы! else располагается на одном уровне с if, а условия, выполняемые в случае ложности условия так же имеют отступ от начала строки (то есть выделяется тот блок кода, который будет выполнен в случае, если возраст меньше 18).

**Задания:**

Провести рефакторинг (переделать код) в задаче про Дмитрия и рекламное агентство с использованием оператора else.

**Else if оператор**

Помимо оператора if и оператора else в python существует оператор elif.

Давайте разберемся, как это работает. Предположим, мы хотим сходить в кафе и купить какой-нибудь сэндвич. Мы каждый день туда ходим и четко знаем, что там продается и, конечно, у нас уже есть вкусовые предпочтения. Больше всего мы любим сэндвич с курицей, на втором месте сэндвич с говядиной, на последнем сэндвич с тунцом. В случае отсутствия всех перечисленных возьмем что-то другое. Итак, попробуем написать код с использованием оператора if / else. Введем переменную, содержащую значение того типа сэндвича, который остался в наличии в магазине.

>> sandwich\_in\_stock = ‘sandwich\_

>>if sandwich\_in\_stock == ‘с курицей’:

>> print(‘куплю сэндвич с курицей’)

>>else:

>> if sandwich\_in\_stock == ‘с говядиной’:

>> print(‘куплю сэндвич с говядиной’)

>> else:

>> if sandwich\_in\_stock == ‘с тунцом’:

>> print(‘куплю сэндвич с тунцом’)

>> else:

>> print(‘возьму что-нибудь другое)

Из этого примера видно, что получилась сложная и сложно читаемая конструкция. Если условие ложно, то внутри оператора else приходится проверять еще дополнительные условия. Это называется вложенные операторы. Обратите внимание, что чем дальше вложен оператор, тем бОльший отступ от начала строки.

Для того, чтобы не прописывать вложенный оператор if в python введен специальный оператор elif.

То есть оператор elif - это именно то, на что похоже «иначе, если». Оператор elif проверяет другое условие после предыдущего условия if, которое не выполняется.

Можно использовать операторы elif для управления порядком, в котором наша программа должна проверять каждый из наших условных операторов. Сначала проверяется оператор if, затем каждый оператор elif проверяется сверху вниз, а затем, наконец, выполняется код else, если ни одно из предыдущих условий не было выполнено.

Давайте переделаем код с сэндвичами и переделаем его с использованием оператора elif.

>>if sandwich\_in\_stock == 'с курицей':

>> print('куплю сэндвич с курицей')

>>elif sandwich\_in\_stock == 'с говядиной':

>> print('куплю сэндвич с говядиной')

>>elif sandwich\_in\_stock == 'с тунцом':

>> print('куплю сэндвич с тунцом')

>>else:

>> print('возьму что-нибудь другое')

Рассмотрим другой пример. Предположим, разрабатывается сайт для сбора пожертвований и в зависимости от того, какую сумму вы жертвуете, вам выводится соответствующее сообщение:

>>if donation >= 1000:

>> print("Спасибо за ваше пожертвование! Вы получили платиновый статус пожертвования!

>>elif donation >= 500:

>> print("Спасибо за ваше пожертвование! Вы получили золотой статус пожертвования!")

>>elif donation >= 100:

>> print("Спасибо за ваше пожертвование! Вы получили серебряный статус пожертвования!")

>>else:

>> print("Спасибо за ваше пожертвование! Вы получили бронзовый статус пожертвования!")")

Подумайте об этой функции на секунду. Что бы произошло, если бы все операторы elif были просто операторами if? Если бы было пожертвовано 1000, то все первые три сообщения были бы выведены в консоль, потому что каждое из условий было выполнено.

Но поскольку мы использовали операторы elif, он последовательно проверяет каждое условие и печатает только одно сообщение. Если я пожертвую 600,00, код сначала проверяет, превышает ли это 1000,00, а это не так, затем он проверяет, превышает ли это 500,00, что так и есть, поэтому выводится соответствующее сообщение, а затем, поскольку все другие утверждения проверяются только при условии ложности предыдущего, то они не проверяются и сообщения больше не выводятся.

**Задание:**

В университете действует система грейдов, которая присваивается студенту в зависимости от среднего балла. Вас просят написать приложение, которое выводило бы соответствующий грейд для каждого студента при следующих условиях:

• 4.0 или выше должен вернуть "A"

• 3.0 или выше должен вернуть "B"

• 2.0 или выше должен вернуть "C"

• 1.0 или выше должен вернуть "D"

• 0,0 или выше должен вернуть "F"

Для решения ввести переменную grade, а также использовать оператор elif.

**Операторы Try и Except**

Операторы if, elif и else - не единственный способ встроить поток управления команд в программу. Существуют операторы try и except для проверки возможных ошибок, с которыми может столкнуться пользователь.

Общий синтаксис операторов try и except:

>>try:

>> # some statement

>>except ErrorName:

>> # some statement

Во-первых, будет выполнен оператор try. Если в какой-то момент во время этого выполнения возникает исключение, такое как NameError или ValueError, и это исключение соответствует ключевому слову в операторе except, тогда оператор try будет завершен, и оператор except будет выполнен.

На самом деле, ошибки можно проверять и с использованием if/else.

Существует 2 подхода к управлению потока:

EAFP (easier to ask forgiveness than it is to get permission, перевод: проще попросить прощения, чем получить разрешение). При этом подходе мы применяем конструкцию try/except. В этом случае при выполнении программы сначала производится попытка что-то сделать (например, осуществить деление), а в случае возникновения ошибки выполняется другой код.

Давайте посмотрим на это на примере. Необходимо написать код, который осуществляет деление a на b. Но есть вероятность, что b равно нулю, что вызовет ошибку, поэтому необходимо включить try и except, чтобы отловить эту ошибку.

>>try:

>> result = a / b

>> print (result)

>>except ZeroDivisionError:

>> print ("деление на ноль не допустимо ")

LBYL (Look before you leap, перевод: смотри, прежде чем прыгать). При этом подходе сначала проверяется, что значения допустимы, а потом выполняется код.

Пример с делением будет выглядеть так:

>>if b!=0:

>> result = a/b

>> print(result)

>>else:

>> print(‘деление на ноль не допустимо’)

**Соответствие структуре шаблона (Structural pattern mapping)**

**Конструкция match\case**

В python 3.10 была введена конструкция match\case. Оператор match похож на оператор switch, присутствующий во многих других языках программирования, и позволяет выполнять ту или иную команду в зависимости от соответствия шаблону. Введение данного оператора в язык python позволило повысить читабельность кода при множестве вариантов шаблонов для объекта.

Общий синтаксис конструкции match/case:

**>>**match subject:

>> case <pattern\_1>:

>> <action\_1>

>> case <pattern\_2>:

>> <action\_2>

>> case <pattern\_3>:

>> <action\_3>

>> case \_:

>> <action\_wildcard>

Например, вернемся к нашему приложению hello world. Теперь мы хотим увеличить функционал этого предложения и выводить приветствие в зависимости от того, какой выбран язык. Это можно тбыло бы сделать с помощью конструкции if/else/elif. Например:

>>if language == ‘русский’:

>> print(‘Привет, мир’)

>>elif language == ‘english’:

>> print(‘Hello, world’)

>>elif language == ‘deutsch’:

>> print(‘Hallo Welt’)

>>else:

>> print(‘unknown language’)

С конструкцией match\case уод будет выглядеть следующим образом:

>>match language:

>> case "russian":

>> print("Hello, world")

>> case "english":

>> print("Hello")

>> case " deutsch":

>> print("Hallo Welt ")

>> case \_:

>> print("unknown language")

Обратите внимание, что блоки case имеют отступы от начала конструкции match. А инструкции каждого блока case имеют отступы от начала данного блока case.

**Задание**

В университете действует система грейдов, которая присваивается студенту в зависимости от среднего балла. Вас просят написать приложение, которое выводило бы соответствующий грейд для каждого студента при следующих условиях:

• 4.0 или выше должен вернуть "A"

• 3.0 или выше должен вернуть "B"

• 2.0 или выше должен вернуть "C"

• 1.0 или выше должен вернуть "D"

• 0,0 или выше должен вернуть "F"

Для решения ввести переменную grade, а также использовать оператор match\case

**Выводы**

В этом разделе мы рассмотрели много новых инструментов языка Python, которые позволяют нам добавить логики в наш код. Подведем итог. В рамках данного раздела было рассмотрено:

• Логические выражения - это утверждения, которые могут иметь значение True или False.

• Логическая переменная - это переменная, для которой установлено значение True или False.

• Вы можете создавать логические выражения, используя операторы сравнения:

o Равно: ==

o Не равно:! =

o Больше чем:>

o Больше или равно:> =

o Меньше чем: <

o Меньше или равно: <=

• операторы if могут использоваться для создания потока управления в вашем коде.

• Операторы else могут использоваться для выполнения кода, когда не выполняются условия оператора if.

• операторы elif могут использоваться для встраивания дополнительных проверок в операторы if.

• Операторы try и except могут использоваться для встраивания контроля ошибок в ваш код.

• Операторы match и case могут использоваться для проверки соответствия шаблону.

Функции

Язык python имеет особенность в построчном исполнении кода. Например, если мы последователь хотим вывести в консоль имя и возраст человека, мы сделаем следующее:

>>name = ‘Анна’

>>age = 18

>>print(name)

>>print(age)

Предположим, что мы хотим вывести несколько разных человек с различными именами и возрастами. Что в таком случае потребуется сделать? Последовательно повторять вышеуказанные строчки кода несколько раз. Например:

>>name = ‘Анна’

>>age = 18

>>print(name)

>>print(age)

>>name = ‘Александр’

>>age = 19

>>print(name)

>>print(age)

>>name = ‘Олег’

>>age = 18

>>print(name)

>>print(age)

Чем больше данных о различных людях мы захотим вывести в консоль, тем больше одинаковых строк появится в коде. Код становится нагроможденным и не читаемым. Как же сделать так, чтобы однотипные операции можно было бы прописать 1 раз, а потом просто исполнять их с разными данными? (в примере операции – это присвоение имени переменным и вывод в консоль, а параметры – непосредственно имена (Анна, Александр, Олег) и возраста (18,19,18))

При решении таких задач в python используются специальные объекты, которые называются функциями.

В данном занятии будет рассмотрено:

* Что есть функция
* Способы задания функций
* Вызовы функций
* Типы параметров функций

**Понятие функции**

Возвращаемся к нашему примеру с именами. Для того, чтобы выделить этот код в отдельную структуру, которая будет срабатывать каждый раз, когда мы хотим вывести в консоль имя и возраст нового человека, мы можем создать функцию и назвать ее printing\_name\_and\_age() и вызывать ее каждый раз при необходимости вывода в консоль информации об имени и возрасте разных людей.

**Как задается функция**

Для того, чтобы создать функцию необходимо использовать ключевое слово def. Далее необходимо задать имя функции. Для имен функций действуют те же правила, что и для имен переменных. После имени функции пишут скобки, а после скобок двоеточие. Вернемся к нашему примеру с выводом имени и возраста в консоль. Создадим функцию printing\_name\_and\_age(), которая будет выводить в консоль нужные нам значения:

>>name = ‘Анна’

>>age = 18

>>def printing\_name\_and\_age():

>> print(name)

>> print(age)

Как видно из примера, та часть кода, которая включена в функцию, имеет отступ от края строки (как в случае с if-else). Для того, чтобы код в функции исполнился, необходимо ее вызвать. Для вызова функции необходимо написать ее название. В нашем примере:

>>name = ‘Анна’

>>age = 18

>>def printing\_name\_and\_age():

>> print(name)

>> print(age)

>> printing\_name\_and\_age()

Последняя строка – это вызов функции. Если не осуществить вызов функции, то код, находящийся внутри нее (все, что с отступами после двоеточия) не будет исполнен. С использованием функций наш код преобразуется в следующий:

>>def printing\_name\_and\_age():

>> print(name)

>> print(age)

>>name = ‘Анна’

>>age = 18

>> printing\_name\_and\_age()

>>name = ‘Александр’

>>age = 19

>> printing\_name\_and\_age()

>>name = ‘Олег’

>>age = 18

>> printing\_name\_and\_age()

Как видим, у нас сократилось количество повторяющихся строк. Однако, нам бы хотелось не назначать каждый раз значения переменным name и age, а просто вызывать ее с разными параметрами.

**Входные параметры функции**

В python есть возможность задавать входные параметры функций. Входные параметры записываются в скобках после названия функции и отделяются запятой. Вернемся к нашему примеру. Мы хотим, чтобы функция def printing\_name\_and\_age() принимала в качестве входных параметров имя и возраст людей, а потом выводила их в консоль. С использованием параметров функции это будет выглядеть следующим образом:

>>def printing\_name\_and\_age(name, age):

>> print(name)

>> print(age)

Значения этих параметров будет задаваться при вызове функции внутри скобок. Для вывода данных из более раннего примера код будет выглядеть следующим образом:

>>def printing\_name\_and\_age(name, age):

>> print(name)

>> print(age)

>>printing\_name\_and\_age(‘Анна’, 18)

>>printing\_name\_and\_age(‘Александр’, 19)

>>printing\_name\_and\_age(‘Олег’, 18)

В консоль будет выведено:

Анна

18

Александр

19

Олег

18

Как видно из примера, при вызове функции в скобках мы указали параметры, которые подставились в соответствующую переменную.

Но как программа узнала, какой параметр куда подставить? На самом деле, значения подставились в параметры функции в порядке их следования. Так как при создании функции мы в скобках написали сначала name, потом age, то ‘Анна’ подставилось в параметр name, как указанное первым в скобках при вызове функции. Если бы был изменен вызов функции, то есть поменяли бы местами параметры ‘Анна’ и 18 так:

>>def printing\_name\_and\_age(name, age):

>> print(name)

>> print(age)

>>printing\_name\_and\_age(18, ‘Анна’)

То вывод бы выглядел так:

18

Анна

Как видно, тогда в name записалось значение 18, а в age значение ‘Анна’. То есть значения параметров должны быть перечислены в вызове функции в том порядке, в котором заданы сами параметры при создании функции. Так как значения параметров при вызове функции задаются в зависимости от позиции параметров при создании функции, такие параметры называют позиционными.

Для вызова функции необходимо в скобках прописывать значения для всех имеющихся в функции параметров. В нашем примере в функции задано 2 параметра name и age. Что будет, если задать значение только для одного из параметров? Например,

>>printing\_name\_and\_age(‘Анна’)

В этом случае будет выведено сообщение об ошибке:

TypeError: printing\_name\_and\_age() missing 1 required positional argument: 'age'

**Ключевые параметры**

Предположим, что мы выводим в консоль информацию о студентах 1-го курса. Мы знаем, что большинству студентов 18 лет. Людей с другим возрастом всего 3 человека. Как быть в этой ситуации?

Для таких случаев существуют параметры, которые имеют значения по умолчанию. То есть мы можем задать значение, которое будет подставляться в соответствующий параметр в случае, если мы его не укажем при вызове функции. Аргументы, имеющие значения по умолчанию, называются ключевыми

В нашем с именами и возрастом можно сделать так:

>>def printing\_name\_and\_age(name, age=18):

>> print(name)

>> print(age)

>>printing\_name\_and\_age(‘Анна’)

Тогда будет выведено:

Анна

18

То есть можно заметить, что при вызове функции возраст мы не указали, однако при создании функции было прописано значение по умолчанию age=18, поэтому не вывелась ошибка, а в консоль было выведено значение 18 в качестве возраста.

Если мы хотим, чтобы возраст был какой-то отличной от значения по умолчанию цифрой, то мы должны указать ее при вызове функции. Например, для вывода данных Александра, которому 19 лет, вывод будет выглядеть так:

>>printing\_name\_and\_age(‘Александр’, 19)

Тогда весь наш исходный код будет выглядеть так:

>>def printing\_name\_and\_age(name, age=18):

>> print(name)

>> print(age)

>>printing\_name\_and\_age(‘Анна’)

>>printing\_name\_and\_age(‘Александр’, 19)

>>printing\_name\_and\_age(‘Олег’)

Необходимо запомнить, что ключевые аргументы, имеющие значения по умолчанию всегда указываются в конце, то есть после позиционных. Поэтому следующий код:

>>def printing\_name\_and\_age(age=18, name):

>> print(name)

>> print(age)

>>printing\_name\_and\_age(‘Анна’)

Выдаст ошибку:

SyntaxError: non-default argument follows default argument

**Задание**

1. Определите функцию с именем create\_spreadsheet (), которая принимает один аргумент title и выводит в консоль только одну строку «Создание электронной таблицы с именем» + title.

2. Вызовите create\_spreadsheet с значением title «Загрузки».

3. Добавьте параметр row\_count в определение функции. Установите значение по умолчанию 1000.

4. Измените оператор print «Создание электронной таблицы с названием title with row\_count lines», где title и row\_count заменяются их соответствующими значениями.

Помните, чтобы объединить число в строковый объект, вам сначала нужно привести row\_count к строке, используя str (). В противном случае вы получите ошибку TypeError.

5. Вызовите create\_spreadsheet () с названием набора в «Приложения» и row\_countset в 10.

**Возвращаемые значения**

В данном занятии будет объяснено, как использовать результаты вычисления функций для их дальнейшего использования для вычислений.

**Оператор return**

До сих пор мы видели только функции, которые выводят какой-то результат в консоль. Функции также могут возвращать значение пользователю, чтобы это значение можно было изменить или использовать позже. Когда результат исполнения функции может быть сохранен в переменной, он называется возвращаемым значением функции. Мы используем ключевое слово return, чтобы вернуть значение функции.

Предположим, у нас есть две функции. Первая складывает 2 числа, вторая вычитает из первого второе. Данные функции реализованы следующим образом:

>>def summing(a, b):

>>result = a+b

>>print(result)

>>def subtraction (a, b):

>>result = a-b

>>print(result)

Предположим, нам нужно вычислить следующее выражение: 2+3-1. Как видно из формулы, мы используем как сложение, так и вычитание последовательно. Вызовем первую функцию сложения и попробуем присвоить значение переменной:

>>x=2

>>y=3

>>z=1

>>result\_sum = summing(x, y)

>>print(result\_sum)

В консоль выведется:

5

None

Мы видим, что result\_sum не содержит результат суммирования, а содержит None. Как поместить в переменную result\_sum результат суммирования? Для этого мы используем оператор return.

>>def summing(a, b):

>>result = a+b

>>return result

>>def subtraction (a, b):

>>result = a-b

>> return result

>>x=2

>>y=3

>>z=1

>>result\_sum = summing(x, y)

>>print(result\_sum)

В результате будет выведено:

5

То есть оператор return позволил поместить значение результата суммирования в переменную result\_sum. Почему нельзя просто вызвать переменную result из функции? Это происходит из-за областей видимости.

**Области видимости**

Допустим, у нас есть функция из последнего упражнения, которая создает строку о специальном элементе:

>>def create\_special\_string(special\_item):

>>return "Our special is " + special\_item + "."

Что если мы хотим получить доступ к переменной special\_item вне функции? Можем ли мы использовать ее?

>>def create\_special\_string(special\_item):

>>return "Our special is " + special\_item + "."

>>print("I don't like " + special\_item)

Если мы попытаемся запустить этот код, мы получим NameError, сообщая нам, что 'special\_item' не определен. Переменная special\_item была определена только внутри пространства функции, поэтому она не существует вне функции. Область действия special\_item - только функция create\_special\_string.

Переменные, определенные вне области действия функции, могут быть доступны внутри тела функции:

>>header\_string = "Our special is "

>>def create\_special\_string(special\_item):

>>return header\_string + special\_item + "."

>>print(create\_special\_string("grapes"))

Здесь нет ошибки. header\_string может использоваться внутри функции create\_special\_string, потому что область действия header\_string - весь файл. Этот файл будет выводить:

Our special is grapes.

**Задание**

1. Функция define\_age создает переменную с именем age, которая представляет собой разницу между текущим годом и годом рождения, оба из которых являются входными данными для функции. Добавьте строку, чтобы возраст возвращался в качестве результата вычислений для дальнейшего использования.

>>def calc\_age (current\_year, birth\_year):

#возраст = текущий\_год - рождение\_год

2. Вне функции вызовите Calculate\_age со значениями 2049 (current\_year) и 1993 (birth\_year) и сохраните значение в переменную my\_age.

3. Вызовите Calculate\_age со значениями 2049 (current\_year) и 1953 (birth\_year) и сохраните значение в переменной с именем dads\_age.

Выведите на консоль строку «Мне X лет, а моему отцу Y лет», с my\_age, где X и dads\_age, где Y.

**Несколько возвращаемых значений**

Иногда может понадобиться вернуть более одного значения из функции. Мы можем вернуть несколько значений, разделив их запятой:

>>def square\_point(x\_value, y\_value):

>> x\_2 = x\_value \* x\_value

>> y\_2 = y\_value \* y\_value

>> return x\_2, y\_2

Эта функция принимает значение x и значение y и возвращает их оба в квадрате. Мы можем получить эти значения, указав их при вызове функции:

>>x\_squared, y\_squared = square\_point(1, 3)

>>print(x\_squared)

>>print(y\_squared)

В консоль выведется:

1

9

**Задание**

1. Напишите функцию с именем get\_boundaries (), которая принимает два параметра: числовой параметр target и числовой параметр margin.

Следует создать две переменные:

• low\_limit: target минус margin.

• high\_limit: margin прибавить к target

2.Возвратить значения low\_limit и high\_limit из функции в указанном порядке.

3.Вызовите функцию с параметром target, равным 100 и с margin 20. Сохраните возвращаемые значения в переменные, называемые low\_limit и high\_limit.

4.Выведите в консоль строку:

Нижний предел: low\_limit, верхний предел: high\_limit

Со значениями low и high, которые вы получили из функции get\_boundaries ().

**Задание**

1. Определите функцию с именем repeat\_stuff, которая принимает два входа, stuff и num\_repeats.

Мы хотим, чтобы эта функция вывела в консоль строку с количеством повторений nre\_repeats. Пока только поместите пустой оператор print внутри функции.

1. Вне функции вызовите repeat\_stuff.

Вы можете использовать значение "Row" для stuff и 3 для num\_repeats.

1. Измените оператор вывода внутри repeat\_stuff на returnstatement.

Он должен возвращать stuff \* num\_repeats.

Примечание: умножение строки просто приводит к повторению значения строки! Например:

"na"\*6

приводит к строке "nananananana".

4. Присвойте параметру num\_repeats значение по умолчанию 10.

5. Соедините результат repeat\_stuff ("Row", 3) и строку "Your Boat." вместе и сохраните результат в переменную с именем lyrics.

6. Создайте переменную с именем song и присвойте ей значение repeat\_stuff, вызываемого с только с stuff.

7. Вывести песню в консоль.

Списки

Иногда необходимо объединять несколько объектов в одну структуру. Например, когда вы идете в магазин и вам дают список продуктов для покупки. Как было бы удобно объединить покупки в единый объект?

В данном разделе будет изучено:

* Что такое последовательные типы?
* Что такое список и как с ним работать?
* Методы обработки списков

**Список – это**

Итак, предположим, вы пишите продукты, которые собираетесь купить. Ваш список имеет вид:

1. Хлеб 1 шт
2. Яйца 10 шт
3. Молоко 1 шт
4. Кофе 1 шт

Наименование товара в данном случае будет строкой, а количество штук товара, которое вы намереваетесь купить, число. В настоящий момент было изучено, как задавать переменные. Используя эти знания, список покупок будет выглядеть так:

>>item1 = ‘хлеб’

>>item2 = ‘яйца’

>>item3 = ‘молоко’

>>item4 = ‘кофе’

Если еще и записывать отдельно количество, то это будет еще ряд переменных, в которых будут записаны значения чисел, обозначающих требуемое количество продукта.

Однако это неудобно. Удобнее было бы объединить весь список продуктов вместе и записать в одну переменную. Для этого существует тип данных list (список). Списки относятся к последовательному типу данных (то есть хранит последовательность элементов) и представляет собой набор отдельных элементов, отделенных друг от друга запятыми. Список обозначается квадратными скобками [ ]. Для нашего примера с покупками список будет иметь вид:

>>shopping\_list = [‘хлеб’, ‘яйца’, ‘молоко’, ‘кофе’]

Как видим из примера, shopping\_list состоит из набора строк, обозначающими наименование отдельных товаров из списка покупок. Однако, так же необходимо записать не только наименования товаров, но и их количество. Возможно создать отдельный список:

>>quantity = [1, 10, 1, 1]

До этого мы рассмотрели два списка, которые состояли из объектов одного типа данных. Список shopping\_list содержит только строки, а quantity только целые числа. Можно вспомнить, что python динамически типизированный язык. Поэтому список может содержать элементы разных типов:

>>bread = [‘хлеб’, 1]

**Задание**

1. Создайте список, который содержит наименование товара «торт» и цифру 1.

**Список списков**

Мы выяснили, что элементами списков могут быть любой тип. На самом деле список может содержать другие списки.

Вернемся к нашему списку покупок. Мы хотим, чтобы внутри списка хранились наименования товаров и их необходимое количество. Для этих целей можно создать список, который будет включать в себя списки:

>> shopping\_list = [[‘хлеб’, 1], [‘яйца’, 10], [‘молоко’, 1], [‘кофе’, 1]]

**Задание**

1. Нам написали дополнительный список товаров, которые необходимо купить в отделе бытовой химии. Он включает в себя стиральный порошок в количестве 1 штуки и средство для мытья посуды так же 1 штуку. Создайте новый список household\_chemicals, который будет содержать список, состоящий из списков, в которых первым элементом будет наименование товара, а вторым количество.

**Метод Zip**

Вернемся к списку покупок:

1. Хлеб 1 шт
2. Яйца 10 шт
3. Молоко 1 шт
4. Кофе 1 шт

Предположим, что у нас есть два отдельных списка, один включает в себя наименование товаров, второй – их количество:

>>shopping\_list = [‘хлеб’, ‘яйца’, ‘молоко’, ‘кофе’]

>>quantity = [1, 10, 1, 1]

Если бы мы хотели создать список списков, в котором каждое наименование товара было бы вместе с необходимым количеством, мы могли бы использовать команду zip. zip принимает два (или более) списка в качестве входных данных и возвращает объект, содержащий список пар (кортежей). Каждая пара содержит по одному элементу из каждого из входных списков. Однако, команда print не покажет нам результат исполнения функции zip.

>> shopping\_list\_with\_quantity = zip(shopping\_list, quantity)

>>print(shopping\_list\_with\_quantity)

В результате выполнения кода выше в консоль будет выведено следующее:

<zip object at 0x7f1631e86b48>

Эта строка обозначает расположение созданного zip объекта в памяти. Однако, у нас есть операция преобразования типов. Ранее мы сталкивались с такой функций, как str(), которая переводила тип данных в строку. Такая же функция существует для преобразования типа данных в список. Эта функция list(). В нашем примере можно использовать данную функцию, чтобы увидеть результат выполнения функции zip:

>> shopping\_list\_with\_quantity = zip(shopping\_list, quantity)

>>print(list(shopping\_list\_with\_quantity))

Результатом исполнения этого кода будет:

[('хлеб', 1), ('яйца', 10), ('молоко', 1), ('кофе', 1)]

**Задание**

1. Используйте zip для создания новой переменной с именем names\_and\_dogs\_names, которая объединяет имена и имена собак в zip-объект.

Names=[‘Ben’, ‘Holly’, ‘Ann’]

dogs\_names= [‘Sharik’, ‘Gab’, ‘Beethoven’]

1. Создайте новую переменную с именем list\_of\_names\_and\_dogs\_names, вызвав list () для names\_and\_dogs\_names. Выведите в консоль новую переменную.

**Пустые списки**

Список не обязательно должен содержать что-то! Вы можете создать пустой список следующим образом:

>>empty\_list = []

Зачем необходим пустой список?

Обычно, такие списки создаются с целью заполнения его позже на основе новых данных.

**Метод append**

Данный метод позволяет добавлять элементы в конец списка. Например, нам нужно купить еще и шоколадку, и мы хотим отразить это в списке покупок. Для этого сделаем следующее:

>>shopping\_list = [‘хлеб’, ‘яйца’, ‘молоко’, ‘кофе’]

>> shopping\_list.append(‘шоколадка’)

>>print (shopping\_list)

Этот код выведет следующее:

['хлеб', 'яйца', 'молоко', 'кофе', 'шоколадка']

Этот метод работает и с пустыми списками. Например,

>>empty\_list = []

>>empty\_list.append(1)

>>print(empty\_list)

Этот код выведет:

[1]

Важно помнить, что .append () идет после списка. Это отличается от функций наподобие print, которые были изучены раньше.

**Задание**

1. Мария работает в цветочном магазине. Она ведет учет заказов в списке, который называется orders.

Создайте список orders и используйте print, чтобы проверить заказы, которые он получил сегодня.

orders = ['маргаритки', 'васильки'].

2. Мария только что получила новый заказ на тюльпаны. Используйте append, чтобы добавить эту строку в orders.

3. Пришел еще один заказ! Используйте append, чтобы добавлять «розы» к заказам.

4. Используйте print, чтобы просмотреть заказы, которые получила сегодня Мария.

**Оператор + при работе со списками**

Ранее было рассмотрено, как работает оператор + с числовыми и строковыми типами данных. Для списков данный оператор так же определен. + позволяет объединять несколько списков вместе. Например, пекарня имеет следующий ассортимент:

>>items\_sold = ['cake', 'cookie', 'bread']

Предположим, пекарня хочет начать продавать печенье и пирожные:

>>items\_sold\_new = items\_sold + ['biscuit', 'tart']

>>print(items\_sold\_new)

['cake', 'cookie', 'bread', 'biscuit', 'tart']

В этом примере мы создали новую переменную items\_sold\_new, которая содержала как оригинальные проданные товары, так и новые. Мы можем проверить исходный items\_sold и убедиться, что он не изменился:

>>print(items\_sold)

['cake', 'cookie', 'bread']

Так же, как и в случае со строками и числами, + можно применять только к спискам. Если мы введем этот код:

>>my\_list = [1, 2, 3]

>>my\_list + 4

мы получим следующую ошибку:

TypeError: can only concatenate list (not "int") to list

Если мы хотим добавить один элемент с помощью +, мы должны поместить его в список с помощью скобок ([]):

>>my\_list + [4]

**Задание**

1. Мария все еще обновляет свой список заказов. Она только что получила заказы на «сирень» и «ирис».

Используйте +, чтобы создать новый список с именем new\_orders, который объединяет заказы с двумя новыми заказами.

orders = ['маргаритка', 'лютик', 'львиный зев', 'гардения', 'лилия']

# broken\_prices = [5, 3, 4, 5, 4] + 4

1. Удалите # перед списком broken\_prices. Если вы запустите этот код, вы получите сообщение об ошибке
2. Исправьте команду, чтобы она выполнялась без ошибок.

**Range I**

Иногда нам необходимо создавать списки, состоящие из последовательностей. Например, необходимо создать список вида:

>>my\_list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Набор такого числа чисел занимает определенное время. Кроме того, чем больше чисел мы набираем, тем больше вероятность того, что может произойти опечатка.

Python дает нам простой способ создания этих списков с помощью функции, называемой range. range принимает число на вход и генерирует числа, начиная с 0 и заканчивая числом перед входным параметром. Итак, если нам нужны числа от 0 до 9, мы используем диапазон (10), потому что 10 на 1 больше 9:

>>my\_range = range(10)

Как и в случае с zip, функция диапазона возвращает объект, который мы можем преобразовать в список:

>>range(0, 10)

>>print(list(my\_range))

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

**Задание**

1. Измените list1 так, чтобы это был диапазон, содержащий числа, начинающиеся с 0 и до 9, но не включая 9.

list1=[1,8]

1. Создайте диапазон с именем list2 с числами от 0 до 7.

**Range II**

Мы можем использовать range для создания более сложных последовательностей.

По умолчанию диапазон создает список, начинающийся с 0. Однако, если мы прописать в качестве входных параметров 2 числа, то можно задать начало и конец создаваемой последовательности. Например, диапазон (2, 9) будет генерировать числа, начиная с 2 и заканчивая 8 (не включая 9):

>> my\_list = range(2, 9)

>>>print(list(my\_list))

[2, 3, 4, 5, 6, 7, 8]

С одним или двумя аргументами range создаст список последовательных чисел (т.е. каждое число на единицу больше предыдущего). Если мы используем третий аргумент, мы можем создать список, который «пропускает» числа. Например, диапазон (2, 9, 2) даст нам список, в котором каждое число на 2 больше предыдущего:

>>> my\_range2 = range(2, 9, 2)

>>> print(list(my\_range2))

[2, 4, 6, 8]

Мы можем пропустить столько чисел, сколько захотим! В этом примере мы начнем с 1 и пропустим 10 между каждым числом, пока не дойдем до 100:

>>my\_range3 = range(1, 100, 10)

>>print(list(my\_range3))

[1, 11, 21, 31, 41, 51, 61, 71, 81, 91]

Наш список останавливается на 91, потому что следующим числом в последовательности будет 101, что больше 100 (наша точка остановки).

Значения аргументов функции range можно обозначить, как (stert, stop, step) (начало, конец, шаг).

**Задание**

1. Измените функцию диапазона, создавшую list1, так, чтобы она:

• Начиналась с 5

• Разница между каждым элементом составляет 3 единицы.

• Заканчивается на 15

list1 = диапазон (6, 15, 2)

1. Создайте объект диапазона с именем list2, который:

• Начинается с 0

• Разница между каждым элементом составляет 5 единиц.

• Заканчивается до 40

**Заключение**

В данном занятии было изучено:

• Как создать список

• Как создать список списков с помощью zip-архива.

• Как добавлять элементы в список с помощью .append () или +

• Как использовать диапазон для создания списков целых чисел

**Задания**

1. Мария вводит данные клиентов для своего бизнеса в области веб-дизайна. Вы поможете ей организовать свои данные.

Начните с превращения этого списка имен клиентов в список с именем first\_names. Обязательно вводите имена в таком порядке:

• Анна

• Борис

• Александр

• Денис

2. Создайте пустой список под названием age.

3. Возраст Дениса - 42 года. Используйте .append (), чтобы прибавить 42 к age.

4. Марии нужен список возрастов для всех клиентов. Создайте новый список с именем all\_ages, который добавляет возраст со следующим списком, содержащим возраст Анны, Бориса и Александра:

[32, 41, 29]

Убедитесь, что all\_ages начинается с возраста Анны, Бориса и Александра и заканчивается возрастом Дениса (хранится в age)!

5. Создайте новую переменную name\_and\_age, которая объединяет first\_names и all\_ages с помощью zip.

6. Создайте диапазон с именем ids с идентификационным номером для каждого клиента. Поскольку клиентов 4, значения id должны изменяться от 0 до 3.

**Операции со списками**

Ранее было изучено, как создавать списки и добавлять в них элементы.

В данном занятии будет изучено:

• Функция получения длины списка.

• Выбор подмножеств списка (называется срез).

• Возможность расчета, сколько раз элемент появляется в списке.

• Сортировка список предметов

**Длина списка**

Часто нам нужно найти количество элементов в списке, обычно называемое его длиной.

Мы можем сделать это с помощью функции len. Когда мы применяем len к списку, мы получаем количество элементов в этом списке:

>>my\_list = [1, 2, 3, 4, 5]

>>print(len(my\_list))

Результатом данного кода будет::

5

**Задание**

1. Вычислите длину list1 и сохраните ее в переменной list1\_len.

list1 = range (2, 20, 2)

2. Используйте print для проверки list1\_len.

3. Измените команду range, которая генерирует list1, чтобы она пропускала 3 вместо 2 элемента.

Как это изменит list1\_len?

**Выбор элементов списка I**

Крис проводит собеседование с кандидатами на работу. Он будет называть каждого кандидата по порядку, представленному списком Python:

>>calls = ['Алиса', 'Иван', 'Сергей', 'Денис', 'Анна']

Сначала он позвонит «Алисе», затем «Иван» и т. д

В Python порядок элемента в списке называют его индексом.

Списки Python начинаются с нулевого индекса. Это означает, что первый элемент в списке имеет индекс 0, а не 1.

Вот порядковые номера этого списка:

| **Element** | **Index** |
| --- | --- |
| ' Алиса' | 0 |
| ' Иван' | 1 |
| ' Сергей' | 2 |
| ' Денис' | 3 |
| ' Анна' | 4 |

В этом примере элемент с индексом 2 - «Сергей».

Мы можем выбрать один элемент из списка, используя квадратные скобки ([]) и индекс элемента списка. Например, если бы мы хотели выбрать третий элемент из списка, мы бы использовали вызовы [2]:

>>print(calls[2])

'Сергей'

1. Используйте квадратные скобки ([ ]), чтобы выбрать элемент с индексом 4 из списка сотрудников. Сохраните его в переменной index4.

employees = ['Майкл', 'Дуайт', 'Джим', 'Пэм', 'Райан', 'Энди', 'Роберт']

1. Используйте print и len, чтобы отобразить, сколько элементов находится у сотрудников.
2. Вставьте в код следующую команду:

print(employees[8])

1. При выборе несуществующего элемента возникает ошибка IndexError.

В строке кода, которую вы вставили, измените 8 на другое число, чтобы не получить IndexError.

**Выбор элементов списка II**

Что, если мы хотим выбрать последний элемент списка?

Для этого в python существуют отрицательные индексы. Можно использовать индекс -1, чтобы выбрать последний элемент списка.

Рассмотрим следующий список из 5 элементов:

>>list1 = ['a', 'b', 'c', 'd', 'e']

Если мы выберем элемент -1, мы получим последний элемент, 'e':

>>print(list1[-1])

'e'

Это то же самое, что и выбор элемента с индексом 4:

>>> print(list1[4])

'e'

**Задание**

1. Используйте print и len, чтобы отобразить длину shopping\_list.

shopping\_list = ['яйца', 'масло', 'молоко', 'огурцы', 'сок', 'хлопья']

2. Получите последний элемент shopping\_list, используя индекс -1. Сохраните этот элемент в переменной last\_element.

3. Теперь выберите элемент с индексом 5 и сохраните его в переменной element5.

4. Используйте print для отображения как element5, так и last\_element.

**Срезы списков**

Допустим, у нас есть список букв:

>>letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g']

Предположим, мы хотим выбрать от b до f.

Мы можем сделать это, используя следующий синтаксис: буквы [начало: конец], где:

• start - это индекс первого элемента, который мы хотим включить в наш выбор. В этом случае мы хотим начать с b, у которого индекс 1.

• end - это индекс на один больше, чем последний индекс, который мы хотим включить. Последний элемент, который нам нужен, - это f с индексом 5, поэтому end должно быть 6.

>>sublist = letters[1:6]

>>print(sublist)

Этот код выведет:

['b', 'c', 'd', 'e', 'f']

Обратите внимание, что элемент с индексом 6 (то есть g) не включен в наш выбор.

**Задание**

1. Используйте print, чтобы проверить переменную beginning.

suitcase = ['рубашка', 'рубашка', 'брюки', 'брюки', 'пижамы', 'книги']

beginning = suitcase [0: 2]

Сколько элементов в списке?

2. Измените beginning, чтобы оно выделяло первые 4 элемента чемодана.

3. Создайте новый список под названием middle, содержащий два средних элемента из чемодана.

**Срез списков II**

Если мы хотим выбрать первые 3 элемента списка, мы могли бы использовать следующий код:

>>fruits = ['яблоко', 'банан', вишня', 'дыня']

>>print(fruits[0:3])

['яблоко', 'банан', 'вишня']

Начиная с начала списка, также допустимо опускать 0:

>>print(fruits[:3])

['яблоко', 'банан', 'вишня']

Мы можем сделать нечто подобное при выборе нескольких последних элементов списка:

>>print(fruits[2:])

['вишня' , 'дыня']

Можно опустить последний индекс при выборе последних элементов из списка.

Если необходимо выбрать последние 3 элемента фруктов, мы также можем использовать этот синтаксис:

>>print(fruits[-3:])

['банан', 'вишня', 'дыня']

Допустимо использовать отрицательные индексы для обратного отсчета от последнего элемента.

**Задание**

Создайте новый список под названием start, содержащий первые 3 элемента чемодана.

suitcase = ['рубашка', 'футболка', 'носки', 'очки', 'пижама', 'книги']

**Подсчет элементов в списке**

Предположим, у нас есть список букв, который представляет буквы в слове «Миссисипи»:

>>letters = ['м', 'и', 'с', 'с', 'и', 'с', 'и', 'п', 'и']

Если мы хотим знать, сколько раз «и» встречается в этом слове, мы можем использовать функцию count:

>>num\_i = letters.count('i')

>>print(num\_i)

В результате исполнения этого кода будет выведено:

4

**Задание**

Класс миссис Уилсон голосует за президента класса. Она сохранила голос каждого студента в списке голосов.

Используйте счетчик, чтобы определить, сколько студентов проголосовало за «Jake». Сохраните свой ответ как jake\_votes.

votes = ['Jake', 'Jake', 'Laurie', 'Laurie', 'Laurie', 'Jake', 'Jake', 'Jake', 'Laurie', 'Cassie', 'Cassie', 'Jake', 'Jake', 'Cassie', 'Laurie', 'Cassie', 'Jake', 'Jake', 'Cassie', 'Laurie']

Используйте print для проверки jake\_votes.

**Сортировка списков I**

Иногда нам нужно отсортировать список по возрастанию (1, 2, 3,…) или в алфавитном (a, b, c,…) порядке.

Мы можем отсортировать список, используя .sort (). Предположим, у нас есть список имен:

>>names = ['Xander', 'Buffy', 'Angel', 'Willow', 'Giles']

>>print(names)

Это выведет:

['Xander', 'Buffy', 'Angel', 'Willow', 'Giles']

Теперь применим .sort ():

>>names.sort()

>>print(names)

И получаем:

['Angel', 'Buffy', 'Giles', 'Willow', 'Xander']

Обратите внимание, что функция сортировки пишется после списка имен. Если попробовать sort (names), мы получим NameError.

sort ничего не возвращает. Итак, если мы попытаемся присвоить переменной names.sort (), наша новая переменная будет None:

>>sorted\_names = names.sort()

>>print(sorted\_names)

это вернет:

None

Хотя sorted\_names имеет значение None, строка sorted\_names = names.sort () по-прежнему редактирует имена:

>>print(names)

['Angel', 'Buffy', 'Giles', 'Willow', 'Xander']

**Задание**

Отсортируйте адреса.

addresses = ['221 B Baker St.', '42 Wallaby Way', '12 Grimmauld Place', '742 Evergreen Terrace', '1600 Pennsylvania Ave', '10 Downing St.']

Используйте print, чтобы увидеть, как менялись addresses

**Сортировка списков II**

Второй способ сортировки списка - использовать sorted. sorted отличается от .sort () следующим:

1. Оно идет перед списком, а не после него.

2. Создает новый список.

Вернемся к нашему списку имен:

>>names = ['Xander', 'Buffy', 'Angel', 'Willow', 'Giles']

Используя sorted, мы можем создать новый список с именем sorted\_names:

>>sorted\_names = sorted(names)

>>print(sorted\_names)

Это дает список, отсортированный по алфавиту:

['Angel', 'Buffy', 'Giles', 'Willow', 'Xander']

Обратите внимание, что использование sorted не изменило names:

>>print(names)

['Xander', 'Buffy', 'Angel', 'Willow', 'Giles']

**Задание**

1. Используйте сортировку, чтобы упорядочить игры и создать новый список с именем games\_sorted.

games = ['Portal', 'Minecraft', 'Pacman', 'Tetris', 'The Sims', 'Pokemon']

1. Используйте print для проверки игр и games\_sorted.

**Выводы**

Было изучено:

• Как получить длину списка

• Выбор подмножеств списка (называется срезом).

• Расчет того, сколько раз элемент появляется в списке.

• Сортировка списка элементов

**Задание**

1. inventory - это список предметов, которые есть на складе мебели. Сколько товаров на складе?

Сохраните свой ответ в inventory\_len.

inventory = [«двухспальная кровать», «двухспальная кровать», «изголовье», «двуспальная кровать», «двуспальная кровать», «комод», «комод», «стол», «стол», «тумбочка», «тумбочка», «королевский кровать», «двуспальная кровать», «две односпальные кровати», «две односпальные кровати», «простыни», «простыни», «подушка», «подушка»]

2. Выберите первый элемент в инвентаре. Сначала сохраните его в переменной first.

3. Выберите последний элемент из инвентаря и сохраните его в переменной last.

4. Выберите предметы из инвентаря, начиная с индекса 2 и до индекса 6, но не включая его.

Сохраните свой ответ в inventory\_2\_6.

1. Выберите первые 3 предмета инвентаря и сохраните их в переменной first\_3.
2. Сколько односпальных кроватей в инвентаре? Сохраните свой ответ в twin\_beds.
3. Сортировка инвентаря с помощью .sort ().

**Методы работы со списками**

В рамках данного будут изучены методы:

* extend ()
* insert()
* remove()
* pop()
* clear()
* index()
* reverse()
* copy()

**Расширение списка**

Предположим, у нас есть список дел, которые мы хотим сегодня сделать:

>> to\_do\_list = ['приготовить завтрак', 'поставить стирку', 'позвонить клиенту',

'приготовить ужин', 'помыть посуду']

И тут мы обнаружили, что перед сном мы хотели повесить выстиранное белье, но совершенно забыли внести его в список дел! Для того, чтобы добавить список дополнительными значениями существует несколько методов:

1. Использовать оператор + для добавления в конец списка

>>to\_do\_list = ['приготовить завтрак', 'поставить стирку', 'позвонить клиенту', 'приготовить ужин', 'помыть посуду']

>>to\_do\_list+=['повесить белье']

>>print(to\_do\_list)

Результатом будет:

['приготовить завтрак', 'поставить стирку', 'позвонить клиенту', 'приготовить ужин', 'помыть посуду', 'повесить белье']

1. Использовать срез и вычисление длины списка

>>to\_do\_list = ['приготовить завтрак', 'поставить стирку', 'позвонить клиенту', 'приготовить ужин', 'помыть посуду']

>>to\_do\_list[len(to\_do\_list):]=['повесить белье']

>>print(to\_do\_list)

Результат аналогичен предыдущему примеру.

В данном примере была рассчитана длина списка, а далее указано, что начиная с элемента с индексом, равным текущему значению длины и до конца вставить значения другого списка. В результате, в конец списка были добавлены значения другого списка

1. Использовать метод extend()

>>to\_do\_list = ['приготовить завтрак', 'поставить стирку', 'позвонить клиенту', 'приготовить ужин', 'помыть посуду']

>>to\_do\_list.extend('повесить белье')

>>print(to\_do\_list)

Результат аналогичен предыдущему примеру.

Метод extend позволяет так же добавлять элементы внутри списка. Использовать можно любой из представленных методов. Главное не забывать, что данный метод принимает только последовательные типы данных.

**Задание**

1. Марк работает официантом в ресторане. Он принял заказ и внес его в список. Создайте список order и внесите в него пасту, пиццу и салат капрезе.
2. К столику присоединился еще один посетитель и Марк принял у него заказ, состоящий из салата цезарь и кофе. Добавьте в конец списка order данные позиции
3. Клиенты вспомнили, что совсем забыли про напитки! Добавьте в список заказа красное сухое вино.

**Метод insert**

Иногда возникает необходимость внести элемент не в конец списка, а в определенную позицию, обозначенную индексом. Для этого существует метод insert(). Данный метод принимает 2 аргумента:

* Позиция (индекс), на которую требуется поместить элемент. Нумерация ведётся с нуля. Поддерживается отрицательная индексация.
* Элемент, который требуется поместить в список.

Например, мы вспомнили, что забыли помыть посуду после завтрака! Необходимо внести данное дело в список после пункта «Приготовить завтрак». Для этого сделаем следующее:

>>to\_do\_list = ['приготовить завтрак', 'поставить стирку', 'позвонить клиенту',

'приготовить ужин', 'помыть посуду']

>>to\_do\_list.insert(1, 'помыть посуду')

>>print(to\_do\_list)

В результате в консоль будет выведено:

['приготовить завтрак', 'помыть посуду, 'поставить стирку', 'позвонить клиенту', 'приготовить ужин', 'помыть посуду']

Как видим, в список в позицию под индексом 1 была добавлена необходимая строка.

**Задание**

1. Список с заказом был передан Марком на кухню, однако клиент попросил принести закуску из овощей перед тем, как принесут весь заказ. Добавьте в orders данную позицию в начало списка.

**Методы удаления элементов из списка**

В python существует несколько способов удаления элемента из списка: remove(), pop() и clear(). Помимо них также существует ключевое слово del.

Рассмотрим каждый из них по отдельности.

**Метод remove()**

Данный метод позволяет удалить элемент из списка по значению. Например, мы передумали готовить ужин, так как зашли с коллегами в ресторан.

>>to\_do\_list.insert(1, 'помыть посуду')

>>to\_do\_list.remove('приготовить ужин')

В результате будет выведено:

['приготовить завтрак', 'помыть посуду', 'поставить стирку', 'позвонить клиенту', 'помыть посуду']

Как видим, готовка ужина была удалена из списка дел. Однако осталось мытье посуды после ужина! Но в списке есть 2 пункта про мытье посуды, один после завтрака, один после ужина. Если использовать метод remove(), то будет удален элемент с меньшим значением индекса! В нашем случае:

>>to\_do\_list.remove('приготовить ужин')

Выведет:

['приготовить завтрак', 'поставить стирку', 'позвонить клиенту', 'помыть посуду']

Как мы видим, был удален только пункт, встретившийся первым (индекс 1), а второй элемент «помыть посуду» остался в списке (последний элемент в списке).

Обратите внимание! Если элемента нет, будет выведена ошибка с сообщением о том, что элемент не найден.

**Задание**

1. Клиенты решили, что слишком много еды и решили отказаться от салата капрезе. Удалите данную позицию из списка

**Метод pop()**

Данный метод позволяет удалить элемент из списка по значению индекса. В предыдущем примере мы хотели удалить из списка дел «помыть посуду» (последний элемент в списке). В этом случае можем использовать метод pop():

>>to\_do\_list.pop(-1)

В результате получим:

['приготовить завтрак', 'помыть посуду', 'поставить стирку', 'позвонить клиенту']

Как видим, последний элемент в списке был удален.

Обратите внимание! Если этот индекс не найден или он вне диапазона, то метод выведет исключение IndexError: pop index.

pop() возвращает удаленное значение и позволяет записать его в отдельную переменную. Например, мы хотим поместить в переменную del\_item только что удаленное из списка значение, для этого:

>>del\_item = to\_do\_list.pop(-1)

**Задание**

1. Клиенты Марка вспомнили, что сегодня у них запланирован важный созвон поэтому передумали пить вино. Они попросили Марка удалить этот пункт из заказа. Удалите вино из исходного списка с заказом.

**Метод clear()**

Данный метод позволяет очистить список полностью (удалить все элементы из него). Этот день закончен и мы решили очистить список дел, так как все их выполнили:

>>to\_do\_list.clear()

>>print(to\_do\_list)

В результате получим пустой список:

[]

**Ключевое слово del**

Ключевое слово del так же, как и pop() позволяет удалить элемент из списка по индексу. Однако, оно также дает возможность удалить сразу диапазон значений! Предположим, у нас есть список numbers и мы хотим удалить в нем значения с индексами, начиная от 1 и по 3:

>>numbers = [1, 2, 3, 5, 6, 8, 3]

>>del numbers[1:4]

>>print(numbers)

Данный код выведет:

[1, 6, 8, 3]

Задание

1. Создайте список чисел от 0 до 7
2. Удалите 2 элемента из середины списка, используя del

**Метод reverse()**

Данный метод располагает элементы списка в обратном порядке. Например,

>>my\_list = [1, 'two', 'a', 4]  
>>my\_list.reverse() # None  
>>my\_list # [4, 'a', 'two', 1]

**Задание**

1. Выведите список в обратном порядке

x = [15, 11, 13, 12, 14, 10]

**Метод copy()**

Предположим, у нас есть список lst1 = [0, 1, 2, 3, 4, 5]. Мы хотим создать его копию. Попробуем использовать оператор присваивания и создать переменную lst2, в которой сохраним копию:

>>lst2 = lst1

Предположим, из первого списка мы хотим удалить элемент с индексом 2. Для этого используем изученный сегодня метод pop():

>>lst2 = lst1

>>lst1.pop(2)

>>print(lst2)

>>print(lst1)

Выведем в консоль оба списка. Что мы видим?

[0, 1, 3, 4, 5]

[0, 1, 3, 4, 5]

Как можно заметить, мы удалили элемент с индексом 2 только из одного списка, а удалилось из обоих. Почему так?

Вспомним тему переменные. Оператор присваивания лишь указывает ссылку на объект в памяти. То есть в настоящий момент lst2 и lst1 указывают на одно и тоже место в памяти (ссылаются на один и тот же список).

Для того, чтобы создать полноценную копию списка, необходимо использовать метод copy(), который позволит получить полноценную копию списка удаление из одного списка не повлечет за собой удаление из другого.

>>lst2 = lst1.copy()

>>lst1.pop(2)

>>print(lst2)

>>print(lst1)

Данный код выведет в консоль:

[0, 1, 2, 3, 4, 5]

[0, 1, 3, 4, 5]

Как видно из примера, элемент был удален только из lst1.

Циклы

Представим, что у нас есть список сортов мороженного, которое продается в киоске. Перед нами стоит задача вывести в консоль все эти сорта мороженного по одному. Тогда код будет выглядеть следующим образом:

>>ice\_cream = ['клубничное', 'фисташковое', 'шоколадное', 'малиновое', 'блю курасава', 'лимонное', 'ванильное']

>>print(ice\_cream[0])

>>print(ice\_cream[1])

>>print(ice\_cream [2])

>>print(ice\_cream [3])

>>print(ice\_cream [4])

>>print(ice\_cream [5])

>>print(ice\_cream [6])

Это кажется неэффективным и громоздким. В Python (и большинстве других языках программирования) существует более простой способ перебора (итерации) каждого элемента в списке. Мы можем использовать циклы! Цикл - это способ многократного повторения программного кода.

В этом занятии будет объяснено:

• Циклы, которые позволяют перебирать каждый элемент списка, обозначаемые ключевым словом for.

• Циклы, которые повторяют набранный код, пока не выполнится какое-либо условие, обозначаемые ключевым словом while.

• Циклы, которые создают новые списки, называемые составления списков.

**Создание цикла for**

Возможно вывести каждый элемент списка с помощью цикла for. Цикл for позволяет нам выполнять действие с каждым элементом в списке. Использование каждого элемента списка называется повторением или итерацией (типы данных, к которым применимы циклы, называются итеративными).

Создадим цикл для вывода сортов мороженного из предыдущего примера:

>>ice\_cream = ['клубничное', 'фисташковое', 'шоколадное', 'малиновое', 'блю курасава', 'лимонное', 'ванильное']

>>for item in ice\_cream:

>> print(item)

Общий синтаксис создания цикла for:

>>for <temporary variable> in <list variable>:

>> <action>

В нашем примере с мороженным item была временной переменной, ice\_cream - переменной списка, а print (item) - действием, выполняемым над каждым элементом в списке.

Наша временная переменная может иметь любое имя, и ее не нужно определять заранее. Каждый из следующих фрагментов кода делает то же самое, что и наш пример:

>>for i in ice\_cream:

>> print(i)

>>for ice\_cream\_sort in ice\_cream:

>> print(ice\_cream\_sort)

Обратите внимание, что во всех этих примерах оператор print имеет отступ. Все, что находится на том же уровне отступа после объявления цикла for, включается в цикл for и запускается на каждой итерации.

Если мы забудем сделать отступ, мы получим ошибку IndentationError.

**Задание**

1.Запустите код ниже. Вы должны получить ошибку IndentationError, потому что строка печати (игра) не имеет отступа.

board\_games = ['Settlers of Catan', 'Carcassone', 'Power Grid', 'Agricola', 'Scrabble']

sport\_games = ['football', 'football - American', 'hockey', 'baseball', 'cricket']

for game in board\_games:

print(game)

2.Сделайте отступ в строке 6, чтобы при запуске кода не возникала ошибка IndentationError.

3. Напишите цикл, который выводит каждый вид спорта в sport\_games.

**Использование range в циклах**

Ранее с помощью цикла for перебирались элементы существующего списка.

Иногда существует необходимость перебора не конкретного списка, а выполнения определенного действия несколько раз. Например, если мы хотим вывести в консоль «ВНИМАНИЕ!» сообщение три раза, мы хотели бы сказать что-то вроде:

>>for i in <a list of length 3>:

>> print("ВНИМАНИЕ!")

Обратите внимание, что необходимо перебирать список длиной 3, но нам все равно, что в списке. Чтобы создать эти списки длины n, мы можем использовать функцию range. Range принимает на вход число n и возвращает список от 0 до n-1. Например:

>>zero\_thru\_five = range(6)

>># zero\_thru\_five is now [0, 1, 2, 3, 4, 5]

>>zero\_thru\_one = range(2)

>># zero\_thru\_one is now [0, 1]

Итак, простой способ выполнить наше «ВНИМАНИЕ!» пример будет:

>>for i in range(3):

>> print("ВНИМАНИЕ!")

**Задание**

1. Используйте функцию диапазона в цикле for, чтобы распечатать обещание 5 раз.

promise = "I will not chew gum in class"

**Бесконечные циклы**

В примерах выше осуществлялся перебор элементов таких списков, у которых есть фиксированную длину (начало и конец). Однако давайте рассмотрим этот пример:

>>my\_favorite\_numbers = [4, 8, 15, 16, 42]

>>for number in my\_favorite\_numbers:

>> my\_favorite\_numbers.append(1)

Что здесь происходит? Каждый раз, когда мы входим в цикл, мы добавляем 1 в конец списка, элементы которого мы перебираем. В результате мы никогда не доходим до конца списка! Он продолжает расти!

Цикл, который никогда не завершается, называется бесконечным циклом. Это очень опасно для кода!

Программа, которая попадает в бесконечный цикл, часто становится полностью непригодной для использования. Необходимо избегать бесконечных циклов. Но если вы случайно наткнетесь на один из них, вы можете завершить цикл, нажав Ctrl + c для завершения программы.

**Задание**

1. Предположим, у нас есть два списка студентов: student\_period\_A и student\_period\_B. Мы хотим объединить всех студентов в student\_period\_B.

Напишите цикл for, который проходит через каждого студента в student\_period\_A и добавляет его в конец student\_period\_B.

students\_period\_A = ["Alex", "Briana", "Cheri", "Daniele"]

students\_period\_B = ["Dora", "Minerva", "Alexa", "Obie"]

2. Внутри цикла for, после добавления student к student\_period\_B, выведите student.

3. Предположим, вы допустили опечатку в теле цикла for.

Внутри цикла for измените объект оператора добавления с student\_period\_B на student\_period\_A. В данном случае вами будет получен бесконечный цикл

Выйдите из бесконечного цикла! Затем избавьтесь от ошибки, которая вызвала бесконечный цикл.

**Break**

Мы часто хотим использовать цикл for для поиска некоторого значения в списке:

>>items\_on\_sale = ["blue\_shirt", "striped\_socks", "knit\_dress", "red\_headband", "dinosaur\_onesie"]

>># Мы хотим проверить наличие товара с ID "knit\_dress":

>>for item in items\_on\_sale:

>> if item == "knit\_dress":

>> print("Knit Dress is on sale!")

Этот код просматривает каждый элемент в items\_on\_sale и проверяет соответствие. После того, как мы обнаружим, что knit\_dress находится в списке items\_on\_sale, нам не нужно просматривать остальную часть списка items\_on\_sale. Поскольку он состоит всего из 5 элементов, в этом случае не составляет большого труда перебирать весь список. Но что, если у items\_on\_sale после "knit\_dress" будет 1000 наименований? Что, если бы после "knit\_dress" было 100 000 предметов?

Вы можете остановить цикл for изнутри цикла, используя break. Когда программа встречает оператор break, управление возвращается коду вне цикла for. Например:

>>items\_on\_sale = ["blue\_shirt", "striped\_socks", "knit\_dress", "red\_headband", "dinosaur\_onesie"]

>>print("Checking the sale list!")

>>for item in items\_on\_sale:

>> print(item)

>> if item == "knit\_dress":

>> break

>>print("End of search!")

Это даст результат:

Checking the sale list!

blue\_shirt

striped\_socks

knit\_dress

End of search!

Нам вообще не нужно было проверять "red\_headband" или "dinosaur\_onesie"!

**Continue**

Когда мы перебираем списки, мы можем захотеть пропустить некоторые значения. Допустим, мы хотим распечатать все числа в списке, если они не отрицательные. Мы можем использовать continue для перехода к следующему i в списке:

>>big\_number\_list = [1, 2, -1, 4, -5, 5, 2, -9]

>>for i in big\_number\_list:

>> if i < 0:

>> continue

>> print(i)

Это даст результат:

1

2

4

5

2

**Задание**

1. У вас есть список пород собак, из числа которых вы можете взять себе питомца, dog\_breeds\_available\_for\_adoption. Используя цикл for, выполните итерацию по списку dog\_breeds\_available\_for\_adoption и распечатайте каждую породу собак.

dog\_breeds\_available\_for\_adoption = ['french\_bulldog', 'dalmatian', 'shihtzu', 'poodle', 'collie']

dog\_breed\_I\_want = 'dalmatian'

1. Внутри цикла for проверьте, совпадает ли текущее значение породы с dog\_breed\_I\_want. Если да, то выведите "У них есть собака, которую я хочу!"
2. Добавьте оператор break, когда ваш цикл обнаружит dog\_breed\_I\_want, чтобы не нужно было проверять остальную часть списка.

**While циклы**

Существует другой тип цикла, называемый циклом while. Цикл while выполняет код до тех пор, пока не будет достигнуто какое-либо условие.

Циклы while можно использовать для перебора элементов списков, как и для циклов for:

>>dog\_breeds = ['bulldog', 'dalmation', 'shihtzu', 'poodle', 'collie']

>>index = 0

>>while index < len(dog\_breeds):

>> print(dog\_breeds[index])

>> index += 1

Каждый раз, когда выполняется условие цикла while (в данном случае index <len (dog\_breeds)), выполняется код внутри цикла while.

Циклы while могут быть полезны, когда неизвестно, сколько итераций потребуется для выполнения условия.

**Вложенные циклы**

Мы видели, как можно перебирать элементы списка. Что, если у нас есть список, состоящий из нескольких списков? Как мы можем перебрать все отдельные элементы?

Предположим, мы отвечаем за научный класс, который разделен на три проектные группы:

>>project\_teams = [["Ava", "Samantha", "James"], ["Lucille", "Zed"], ["Edgar", "Gabriel"]]

Если мы хотим получить каждого ученика, мы должны поместить один цикл внутрь другого:

>>for team in project\_teams:

>> for student in team:

>> print(student)

Это приводит к:

Ava

Samantha

James

Lucille

Zed

Edgar

Gabriel

**Задание**

Мы предоставили список sales\_data, который показывает количество различных вкусов мороженого, проданных в трех разных местах вымышленного магазина. Мы хотим суммировать общее количество проданных сортов. Начните с определения переменной scoops\_sold и установите ее равной нулю.

sales\_data = [[12, 17, 22], [2, 10, 3], [5, 12, 13]]

2. Просмотрите список sales\_data. Создайте цикл, перебирающий каждый список из списка sales\_data .

3.В списке sales\_data переберите значения внутри каждого вложенного списка из sales\_data  и прибавьте к своей переменной scoops\_sold.

К концу у вас должна быть сумма всех чисел во вложенном списке sales\_data.

4. Выведите значение scoops\_sold.

**Генераторы списков**

Допустим, мы просмотрели определенный веб-сайт и получили следующие слова:

>>words = ["@coolguy35", "#nofilter", "@kewldawg54", "reply", "timestamp", "@matchamom", "follow", "#updog"]

Мы хотим создать новый список, называемый именами пользователей, в котором будут все строки в словах с символом «@» в качестве первого символа. Мы знаем, что можем сделать это с помощью цикла for:

>>words = ["@coolguy35", "#nofilter", "@kewldawg54", "reply", "timestamp", "@matchamom", "follow", "#updog"]

>>usernames = []

>>for word in words:

>> if word[0] == '@':

>> usernames.append(word)

Сначала мы создали новый пустой список имен пользователей и, просматривая список слов, добавляли каждое слово, соответствующее нашему критерию. Теперь список имен пользователей выглядит так:

>>> print(usernames)

["@coolguy35", "@kewldawg54", "@matchamom"]

В Python есть удобное сокращение для создания подобных списков с одной строкой:

>>usernames = [word for word in words if word[0] == '@']

Это называется генератором списка. Результат будет тот же, что и в цикле for:

["@coolguy35", "@kewldawg54", "@matchamom"]

Понимание этого синтаксиса:

1. Принимает элемент в words

2. Назначает этот элемент переменной с именем word

3. Проверяет, есть ли слово [0] == '@', и если да, добавляет слово в новый список имен пользователей. В противном случае ничего не происходит.

4. Повторите шаги 1–3 для всех строк в словах.

Примечание: если бы мы не проводили никакой проверки (допустим, мы пропустили if word [0] == '@'), новый список был бы просто копией слов:

>>usernames = [word for word in words]

>>#usernames сейчас выглядит так ["@coolguy35", "#nofilter", "@kewldawg54", "reply", "timestamp", "@matchamom", "follow", "#updog"]

Допустим, мы работаем со списком имен пользователей из последнего упражнения:

>>> print(usernames)

["@coolguy35", "@kewldawg54", "@matchamom"]

Мы хотим создать новый список со строкой «пожалуйста, следуй за мной!» добавляется в конец каждого имени пользователя. Мы хотим назвать этот новый список сообщениями. Мы можем использовать понимание списка, чтобы составить этот список из одной строки:

messages = [user + " please follow me!" for user in usernames]

Понимание этого списка:

1. Принимает строку в usernames.

2. Назначает эту строку переменной с именем user.

3. Добавляет «пожалуйста, следуйте за мной!» user

4. Добавляет это объединение в новый список под названием messages.

5. Повторяет шаги 1–4 для всех строк в usernames.

Теперь messages содержат следующие значения:

["@coolguy35 please follow me!", "@kewldawg54 please follow me!", "@matchamom please follow me!"]

Возможность создавать списки с измененными значениями особенно полезна при работе с числами. Допустим, у нас есть этот список:

>>my\_upvotes = [192, 34, 22, 175, 75, 101, 97]

Мы хотим добавить 100 к каждому значению. Мы можем достичь этой цели одной строкой:

>>updated\_upvotes = [vote\_value + 100 for vote\_value in my\_upvotes]

Понимание этого списка:

1. Принимает число в my\_upvotes

2. Присваивает это число переменной с именем vote\_value.

3. Добавляет 100 к vote\_value

4. Добавляет эту сумму в новый список updated\_upvotes.

5. Повторяет шаги 1–4 для всех чисел в my\_upvotes.

Теперь updated\_upvotes содержит следующие значения:

[292, 134, 122, 275, 175, 201, 197]

**Заключение**

На этом уроке вы узнали

• как написать цикл for

• как использовать диапазон в цикле

• что такое бесконечные циклы и как их избежать

• как пропустить значения в цикле

• как написать цикл while

• как составлять списки из одной строки

Давайте еще попрактикуемся с этими концепциями!

**Задание**

1. Создайте список с именем single\_digits, состоящий из чисел от 0 до 9 (включительно).

2. Создайте цикл for, который проходит через single\_digits и выводит каждую из них.

3. Перед циклом создайте список под названием squares. Назначьте для начала пустой список.

4. Внутри цикла, который повторяет single\_digits, добавьте значение квадрата каждого элемента single\_digits к squares. Вы можете сделать это до или после вывода элемента.

5. После цикла for выведите squares.

6. Создайте список cubes, используя сравнение списков в списке single\_digits. Каждый элемент cubes должен быть элементом single\_digits, возведенным в куб.

7. Выведите cubes.

Строки

Слова и предложения являются средством общения, поэтому существует необходимость в том, чтобы компьютеры также имели инструменты работы со словами и предложениями.

В Python мы храним слова, предложения или даже целые абзацы в виде строки. Строка — это последовательность символов. Последовательность может иметь любую длину и содержать любые буквы, цифры, символы и пробелы.

В рамках данного занятия вы узнаете больше о строках и о том, как они обрабатываются в Python. Вы узнаете, как делать срезы строки, выбирать определенные символы из строк, искать в строках символы, выполнять итерацию по строкам и использовать строки в условных операторах.

**Задание**

1. Сохраните ваше любимое слово в виде строки в переменной favour\_word.

2. Выведите favour\_word.

**Это все списки символов!**

Строку можно рассматривать как последовательность символов.

Как и любой другой список, каждый символ в строке имеет индекс. Рассмотрим строку

>>favorite\_fruit = "яблоко"

Мы можем выбрать конкретные буквы из этой строки с помощью индекса. Давайте посмотрим на первую букву строки.

>>> favorite\_fruit[1]

'б'

Обратите внимание, что буква в индексе 1 слова яблоко — это не я, а б. Это связано с тем, что индексы строки начинаются с 0. b находится в нулевом индексе, и мы могли бы выбрать его, используя:

>>> favorite\_fruit[0]

'я'

Важно отметить, что индексы строк должны быть целыми числами. Если бы вы попытались выбрать нецелочисленный индекс, мы бы получили TypeError:

>>> favorite\_fruit[1.5]

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: string indices must be integers, not float

**Как получить срез строки**

Мы можем не только выбрать один символ из строки, мы можем выбрать фрагменты символов из строки. Это возможно сделать это с помощью следующего синтаксиса:

string\_name[first\_index:last\_index]

Это называется срезом строки. Когда мы делаем срез, мы создаем новую строку, которая начинается (и включает) first\_index и заканчивается (но исключает) last\_index. Давайте посмотрим на несколько примеров. Вспомните вашу favorite\_fruit:

>>> favorite\_fruit

'яблоко'

Допустим, нам нужна новая строка, содержащая буквы ок. Мы могли бы сделать срез favorite\_fruit следующим образом:

>>> favorite\_fruit[3:5]

'ок'

Обратите внимание, что символ в первом индексе, я, ВКЛЮЧЕН, но символ в последнем индексе, о, ИСКЛЮЧЕН.

У нас также могут быть не указаны индексы. Если мы удалим первый индекс, срез начнется с начала строки, а если мы удалим второй индекс, срез продолжится до конца строки.

>>> favorite\_fruit[:4]

'ябло'

>>> favorite\_fruit[4:]

'ко'

Опять же, обратите внимание, как к из строки исключен из первого примера и включен во второй пример.

**Задание**

1. Вы программист, работаете в большой компании. В этой компании имя пользователя каждого сотрудника создается путем взятия первых пяти букв их фамилии.

Новый сотрудник, Виталий Красилов, начинает работу сегодня, и вам необходимо создать его учетную запись. Запишите его first\_name и last\_name.

Создайте переменную new\_account, срезав первые пять букв его last\_name.

1. Временные пароли для новых сотрудников также генерируются из их фамилий.

Создайте переменную с именем temp\_password, создав фрагмент с третьей по шестую буквы его last\_name.

**Конкатенация (объединение строк)**

Вы также можете объединить две существующие строки в новую строку. Рассмотрим следующие две строки.

>>fruit\_prefix = "blue"

>>fruit\_suffix = "berries"

Мы можем создать новую строку, объединив их вместе следующим образом:

>>>favorite\_fruit = fruit\_prefix + fruit\_suffix

>>> favorite\_fruit

'blueberries'

Обратите внимание, что здесь не добавлены пробелы. Вы должны вручную добавить пробелы при объединении строк, если хотите их включить.

**Задание**

1. Руководство Компании, в которой вы работаете, осознало, что их политика использования первых пяти букв фамилии сотрудника в качестве имени пользователя не идеальна, если у них несколько сотрудников с одинаковой фамилией.

Напишите функцию с именем account\_generator, которая принимает два входа, first\_name и last\_name, объединяет первые три буквы каждого из них, а затем возвращает новое имя учетной записи.

1. Проверьте свою функцию и сохраните результат ее выполнения в переменную new\_account.

**Измерение длины строки**

Python имеет несколько встроенных функций для работы со строками. Одна из наиболее часто используемых этих функций - len (). len () возвращает количество символов в строке

>>> favorite\_fruit = "blueberry"

>>> len(favorite\_fruit)

9

Если вы измеряете длину предложения, то учитываются и пробелы.

>>> fruit\_sentence = "I love blueberries"

>>> len(fruit\_sentence)

18

Теперь рассмотрим следующий код:

>>> length = len(favorite\_fruit)

>>> favorite\_fruit[length]

Но этот код сгенерирует IndexError, потому что, помните, индексы начинаются с 0, поэтому последний символ в строке имеет индекс len (string\_name) - 1.

>>> favorite\_fruit[length-1]

'y'

Вы также можете вывести последние несколько символов строки с помощью len ():

>>> favorite\_fruit[length-4:]

'erry'

**Задание**

1. Руководство компании снова хочет обновить способ генерации временных паролей для новых сотрудников.

Напишите функцию с именем password\_generator, которая принимает два входа, first\_name и last\_name, а затем объединяет последние три буквы каждого из них и возвращает их в виде строки.

1. Протестируйте свою функцию и сохраните их в переменной temp\_password.

**Отрицательные индексы**

В предыдущем упражнении мы использовали len (), чтобы получить фрагмент символов в конце строки.

Есть более простой способ это сделать - использовать отрицательные индексы! Отрицательные индексы отсчитываются в обратном порядке от конца строки, поэтому string\_name [-1] — это последний символ строки, string\_name [-2] - второй последний символ строки и т. д.

Вот некоторые примеры:

>>> favorite\_fruit = 'blueberry`

>>> favorite\_fruit[-1]

'y'

>>> favorite\_fruit[-2]

'r'

>>> favorite\_fruit[-3:]

'rry'

Обратите внимание, что мы можем вывести последние три символа слова blueberry, имея начальный индекс -3 и опуская конечный индекс.

**Задание**

1. Используйте отрицательные индексы, чтобы найти предпоследний символ в company\_motto. Сохраните это в переменной second\_to\_last.

company\_motto= «Мечты сбываются»

1. Используйте отрицательные индексы, чтобы создать фрагмент из последних 4 символов в company\_motto. Сохраните это в переменной final\_word.

**Строки неизменяемы (иммутабельны)**

До сих пор в этом уроке мы выбирали символы из строк, формировали срезы строки и объединяли строки. Каждый раз, когда мы выполняем одну из этих операций, мы создаем совершенно новую строку.

Это потому, что строки неизменяемы. Это означает, что мы не можем изменить строку после ее создания. Мы можем использовать строку для создания других строк, но мы не можем изменить саму строку.

Это свойство обычно называется неизменяемостью (иммутабельностью). Типы данных, которые являются изменяемыми, могут быть изменены, а типы данных, такие как строки, которые являются неизменяемыми, не могут быть изменены.

**Задание**

1. Последним сотрудником нашей компании стал парень по имени Роб Дейли. К сожалению, отдел кадров, похоже, допустил небольшую опечатку и прислал неправильное имя first\_name.

Попробуйте изменить первый символ first\_name, запустив

first\_name[0] = "Р"

1. Строки неизменяемы, поэтому мы не можем изменить отдельный символ. Хорошо, это не проблема, мы все еще можем это исправить!

Объедините строку «Р» с фрагментом first\_name, который включает все, кроме первого символа, и сохраните его в новой строке fixed\_first\_name.

**Escape-последовательности**

Иногда при работе со строками необходимо включить символы, которые уже имеют особое значение в Python. Например, скажем, создаем строку

>>favorite\_fruit\_conversation = "He said, "blueberries are my favorite!""

Мы случайно закончили строку раньше, чем захотим, включив символ ". Это можно сделать, введя escape-символы. Добавив обратную косую черту перед специальным символом, который мы хотим экранировать, \", мы можем включить его в строку.

>>favorite\_fruit\_conversation = "He said, \"blueberries are my favorite!\""

**Задание**

1. Когда Роб Дейли настраивал свою учетную запись, он установил свой пароль:

theycallme"crazy"91

Его пароль вызывал некоторые ошибки в системе из-за отметок ". Перепишите его пароль, используя escape-символы, и сохраните его с переменным паролем.

**Итерации по строкам**

Теперь вы знаете достаточно о строках, и мы можем начать делать по-настоящему забавные вещи!

Поскольку строки являются последовательностями, это означает, что мы можем перебирать строку, используя циклы for или while. Это открывает целый ряд возможностей, которыми мы можем манипулировать и анализировать строки. Давайте посмотрим на пример.

>>def print\_each\_letter(word):

>> for letter in word:

>> print(letter)

Этот код будет перебирать каждую букву в данном слове и выводить ее в терминал.

>>> favorite\_color = "blue"

>>> print\_each\_letter(favorite\_color)

'b'

'l'

'u'

'e'

**Задание**

1. Давайте воспроизведем уже знакомую вам функцию len ().

Напишите новую функцию с именем get\_length (), которая принимает строку в качестве входных данных и возвращает количество символов в этой строке. Сделаем это, перебирая строку, используя len ()!

**Строки и условные выражения (часть первая)**

Теперь, когда мы можем перебирать строки, мы действительно можем исследовать потенциал строк. Когда мы перебираем символы строки, мы можем производить действия с каждым символом. Если включить условные операторы в эти итерации, мы можем начать делать действительно классные вещи.

Взгляните на следующий код:

>>favorite\_fruit = "blueberry"

>>counter = 0

>>for character in favorite\_fruit:

>> if character == "b":

>> counter = counter + 1

>>print(counter)

Этот код подсчитает количество b в строке «blueberry» (подсказка: их два). Давайте разберемся, что именно делает этот код.

Сначала мы определяем нашу строку, favourite\_fruit, и переменную с именем counter, которую мы устанавливаем равной нулю. Затем цикл for будет перебирать каждый символ в favourite\_fruit и сравнивать его с буквой b.

Каждый раз, когда символ равен b, код увеличивает счетчик переменной на единицу. Затем, как только все символы будут проверены, код выведет переменную-счетчик, сообщая нам, сколько b было в «blueberry». Это отличный пример того, как итерацию по строке можно использовать для решения конкретной задачи, в данном случае подсчитывая количество вхождений определенной буквы в слове.

**Задание**

1. Напишите функцию с именем letter\_check, которая принимает два ввода: слово и букву.

Эта функция должна возвращать True, если слово содержит букву, и False, если нет.

**Строки и условные выражения (часть вторая)**

Есть даже более простой способ, чем перебирать всю строку, чтобы определить, находится ли символ в строке. Мы можем выполнять этот тип проверки более эффективно, используя in. In проверяет, является ли одна строка частью другой строки.

Вот как выглядит синтаксис in:

letter **in** word

Здесь letter in word — это логическое выражение, которое имеет значение Истина, если строковая буква находится в строковом слове. Вот некоторые примеры:

>>>"e" in "blueberry"

True

>>> "a" in "blueberry"

False

Фактически, этот метод более мощный, чем функция, которую вы написали в предыдущем упражнении, потому что он работает не только с буквами, но и со строками целиком.

>>> "blue" in "blueberry"

True

>>> "blue" in "strawberry"

False

**Задание**

1. Напишите функцию с именем contains, которая принимает два аргумента, big\_string и little\_string, и возвращает True, если big\_string содержит little\_string.

Например, contains ("watermelon", "melon») должен возвращать True, а contains ("watermelon", "berry») должен возвращать False.

1. Напишите функцию с именем common\_letters, которая принимает два аргумента, string\_one и string\_two, а затем возвращает список со всеми их общими буквами.

Буквы в возвращаемом списке должны быть уникальными. Например,

common\_letters("banana", "cream")

должен вернуть ['a'].

**Заключение**

Из этого урока вы узнали:

• Строка — это список символов (Последовательный тип данных).

• Символ может быть выбран из строки с помощью его индекса имя\_строки [index]. Эти индексы начинаются с 0.

• «Срез» можно выбрать из строки. Они могут находиться между двумя индексами или могут быть открытыми, выбирая всю строку из точки.

• Строки можно объединять в строки большего размера.

• len () может использоваться для определения количества символов в строке.

• Строки можно перебирать с помощью циклов for.

• Итерация по строкам открывает огромный потенциал для приложений, особенно в сочетании с условными операторами.

**Задание**

1. Руководство компании снова заручилось вашей помощью, чтобы создать функцию для создания имени пользователя и пароля. В этом упражнении вы создадите две функции: username\_generator и password\_generator.

Начнем с username\_generator. Создайте функцию с именем username\_generator, которая принимает два входа, first\_name и last\_name, и возвращает имя пользователя. Имя пользователя должно состоять из первых трех букв их имени и первых четырех букв их фамилии. Если их имя состоит менее чем из трех букв или их фамилия менее четырех букв, следует использовать их имена полностью.

Например, если сотрудника зовут Abe Simpson, функция должна сгенерировать имя пользователя AbeSimp.

1. Теперь что касается временного пароля, они хотят, чтобы функция взяла введенное имя пользователя и сдвинула все буквы на одну вправо, чтобы последняя буква имени пользователя стала первой буквой и так далее. Например, если имя пользователя - AbeSimp, то сгенерированный временный пароль должен быть pAbeSim.

Начните с определения функции password\_generator так, чтобы она принимала один ввод, имя пользователя и определяла в нем пустую строку с именем password.

1. Внутри password\_generator создайте цикл for, который перебирает символы в имени пользователя.

**Методы форматирования**

Есть три строковых метода, которые могут изменить регистр строки. Это .lower (), .upper () и .title ().

• .lower () возвращает строку, содержащую все символы нижнего регистра.

• .upper () возвращает строку, содержащую все символы верхнего регистра.

• .title () возвращает строку в регистре заголовка, что означает, что первая буква каждого слова пишется с заглавной буквы.

Вот пример использования .lower () в действии:

>>> favorite\_song = 'SmOoTH'

>>> favorite\_song\_lowercase = favorite\_song.lower()

>>> favorite\_song\_lowercase

'smooth'

Каждый символ был изменен на нижний регистр. Важно помнить, что строковые методы могут создавать только новые строки, они не изменяют исходную строку.

>>> favorite\_song

'SmOoTH'

Видите, все то же самое! Эти строковые методы отлично подходят для очистки пользовательского ввода и стандартизации форматирования ваших строк.

**Задание**

1. Вы программист, работающий в организации, которая пытается оцифровать и хранить стихи под названием «Preserve the Verse».

Вам дали две строки, название стихотворения и автор, и попросили их немного переформатировать, чтобы они соответствовали правилам базы данных организации.

Сделайте poem\_title заголовком и сохраните его в poem\_title\_fixed.

poem\_title = "spring storm"

poem\_author = "William Carlos Williams"

1. Выведите poem\_title и poem\_title\_fixed.

**Разделение строк**

.split () выполняется для строки, принимает один аргумент и возвращает список подстрок, найденных между данным аргументом (который в случае .split () известен как разделитель). Следует использовать следующий синтаксис:

>>string\_name.split(delimiter)

Если вы не укажете аргумент для .split (), по умолчанию будет выполняться разделение по пробелам.

Например, рассмотрим следующие строки:

>>> man\_its\_a\_hot\_one = "Like seven inches from the midday sun"

>>> man\_its\_a\_hot\_one.split()

['Like', 'seven', 'inches', 'from', 'the', 'midday', 'sun']

.split вернул список с каждым словом в отдельной строке. Важно отметить: если мы запустим .split () для строки без пробелов, мы получим в ответ ту же строку.

**Задание**

1. В переменной строка из первой строки стихотворения Уильяма Карлоса Вильямса «Весенняя буря».

Используйте .split (), чтобы создать список с именем line\_one\_words, который содержит каждое слово в этой строке стихов.

line\_one = "The sky has given over"

Если мы предоставим аргумент для .split (), мы можем указать символ, по которому мы хотим, чтобы наша строка была разделена. Этот аргумент должен быть представлен как сама строка.

Рассмотрим следующий пример:

>>> greatest\_guitarist = "santana"

>>> greatest\_guitarist.split('n')

['sa', 'ta', 'a']

Мы предоставили «n» в качестве аргумента для .split (), поэтому наша строка «santana» была разделена на каждый символ «n» на список из трех строк.

Как вы думаете, что произойдет, если мы разделим одну и ту же строку на «а»?

>>> greatest\_guitarist.split('a')

['s', 'nt', 'n', ' ']

Обратите внимание, что в этом списке есть неожиданная дополнительная строка. Если разделить строку на символ, которым она также заканчивается, в конце списка будет пустая строка.

Вы можете использовать любую строку в качестве аргумента для .split (), что делает его универсальным и мощным инструментом.

**Задание**

1. Ваш босс прислал имена авторов, которые он хочет, чтобы вы подготовили для импорта в базу данных. К сожалению, он отправил их в виде длинной строки с именами, разделенными запятыми.

Используя .split () и указанную строку, создайте список с именем author\_names, содержащий имя каждого отдельного автора в виде собственной строки.

authors = "Audre Lorde, Carlos Williams, Gabriela Mistral, Jean Toomer, An Qi, Walt Whitman, Shel Silverstein, Carmen Boullosa, Kamala Suraiyya, Langston Hughes, Adrienne Rich, Nikki Giovanni"

1. Отличная работа, но теперь оказывается, что им не нужны имена поэтов (почему они просто не сказали это в первый раз !?)
2. Создайте еще один список с именем author\_last\_names, который содержит только фамилии поэтов в предоставленной строке.

**Escape-последовательности**

Мы также можем разделить строки, используя escape-последовательности. Escape-последовательности используются, чтобы указать, что мы хотим разделить что-то в строке, что не обязательно является символом. Мы рассмотрим две escape-последовательности:

• \ n Новая строка

• \ t табуляция

Новая строка или \ n позволит нам разделить многострочную строку по разрывам строки, а \ t позволит нам разделить строку по табуляциям. \ t особенно полезен при работе с определенными наборами данных, потому что нередко строки данных разделяются табуляциями.

Давайте посмотрим на пример разделения с помощью escape-последовательности:

>>smooth\_chorus = \

>>"""And if you said, "This life ain't good enough."

>>I would give my world to lift you up

>>I could change my life to better suit your mood

>>Because you're so smooth"""

>>chorus\_lines = smooth\_chorus.split('\n')

>>print(chorus\_lines)

Этот код разбивает многострочную строку на новые строки (\ n), которые существуют в конце каждой строки, и сохраняет ее в новом списке под названием chorus\_lines. Затем он печатает chorus\_lines, который произведет вывод

['And if you said, "This life ain\'t good enough."', 'I would give my world to lift you up', 'I could change my life to better suit your mood', "Because you're so smooth"]

Новый список содержит каждую строку исходной строки как собственную меньшую строку. Также обратите внимание, что Python автоматически экранировал символ 'при создании нового списка.

**Задание**

1. Организация прислала вам полный текст стихотворения Уильяма Карлоса Уильямса «Весенняя буря». Они хотят, чтобы вы разбили стихотворение на отдельные строки.

Создайте список с именем spring\_storm\_lines, содержащий строку для каждой строки Spring Storm.

spring\_storm\_text = \

"""The sky has given over

its bitterness.

Out of the dark change

all day long

rain falls and falls

as if it would never end.

Still the snow keeps

its hold on the ground.

But water, water

from a thousand runnels!

It collects swiftly,

dappled with black

cuts a way for itself

through green ice in the gutters.

Drop after drop it falls

from the withered grass-stems

of the overhanging embankment."""

**Соединение строк**

Теперь, когда вы научились разбивать строки с помощью .split (), давайте научимся соединять их вместе с помощью .join (). .join () по сути противоположен .split (), он объединяет список строк вместе с заданным разделителем. Синтаксис .join ():

'delimiter'.join(list\_you\_want\_to\_join)

Это может показаться немного странным, потому что с .split () аргумент был разделителем, а теперь аргументом является список. Это связано с тем, что join по-прежнему является строковым методом, а это означает, что он должен воздействовать на строку. Строка в методе .join () является разделителем, к которому вы хотите соединить значения из списка в одну строку, поэтому список, к которому вы хотите присоединиться, должен быть аргументом.

Это может немного сбивать с толку, поэтому давайте рассмотрим пример.

>>> my\_munequita = ['My', 'Spanish', 'Harlem', 'Mona', 'Lisa']

>>> ' '.join(my\_munequita)

Мы берем список строк my\_munequita и объединили его с помощью нашего разделителя '', который представляет собой пробел. Пробел важен, если вы пытаетесь построить предложение из слов, иначе мы бы получили:

'My Spanish Harlem Mona Lisa'

Словари

Словарь — это неупорядоченный набор пар ключ: значение.

Это дает нам возможность сопоставлять данные друг с другом для того, чтобы была возможность быстро находить значения, связанные друг с другом.

Предположим, мы хотим сохранить цены на различные товары, продаваемые в кафе:

* Тост с авокадо - 6 долларов
* Морковный сок - 5 долларов
* Черничный маффин стоит 2 доллара

В Python мы можем создать словарь под названием menu для хранения этих данных:

>>menu = {" Тост с авокадо ": 6, " Морковный сок ": 5, " Черничный маффин": 2}

Обратите внимание:

* Словарь начинается и заканчивается фигурными скобками {и}.
* Каждый элемент состоит из ключа («тост с авокадо») и значения (6).
* Каждая пара ключ: значение разделяется запятой.
* Считается хорошей практикой вставлять пробел после каждой запятой, но код все равно будет работать и без пробела.

**Задание**

1. Предположим, у нас есть словарь датчиков температуры в доме и текущие значения температуры. Мы только что добавили датчик в «кладовую», он показывает 22 градуса.

Добавьте эту пару в словарь:

sensors =  {"living room": 21, "kitchen": 23, "bedroom": 20}

1. Скопируйте строчку кода ниже и удалите символ # перед определением словаря num\_cameras, который представляет количество камер в каждой области вокруг дома.

Если вы запустите этот код, вы получите сообщение об ошибке:

SyntaxError: invalid syntax

Попробуйте найти и исправить синтаксическую ошибку, чтобы запустить этот код.

#num\_cameras = {"backyard": 6,  "garage": 2 "driveway" 1}

**Создание словаря**

В предыдущем упражнении мы видели словарь, в котором сопоставлены строки и числа (например, «тост с авокадо»: 6). Однако ключи могут быть и числами.

Например, если мы сопоставляем промежуточные итоги счетов в ресторане с общей суммой счетов после чаевых, словарь мог бы выглядеть так:

>>subtotal\_to\_total = {20: 24, 10: 12, 5: 6, 15: 18}

Значения могут быть любого типа. Мы можем использовать строку, число, список или даже другой словарь в качестве значения, связанного с ключом!

Например:

>>students\_in\_classes = {"software design": ["Aaron", "Delila", "Samson"], "cartography": ["Christopher", "Juan", "Marco"], "philosophy": ["Frederica", "Manuel"]}

Список ["Aaron", "Delila", "Samson"], который является значением ключа «software design», представляет учащихся этого класса.

Мы также можем смешивать и сопоставлять типы ключей и значений. Например:

>>person = {"name": "Shuri", "age": 18, "family": ["T'Chaka", "Ramonda"]}

**Задание**

1. Создайте словарь переводов, который сопоставляет следующие слова на английском языке с их определениями на синдарине (языке эльфов):

| **English** | **Sindarin** |
| --- | --- |
| mountain | orod |
| bread | bass |
| friend | mellon |
| horse | roch |

**Недействительные ключи**

У нас может быть список или словарь как значение элемента в словаре, но мы не можем использовать эти типы данных в качестве ключей словаря. Если мы попытаемся это сделать, мы получим TypeError.

Например:

>>powers = {[1, 2, 4, 8, 16]: 2, [1, 3, 9, 27, 81]: 3}

Этот код даст:

TypeError: unhashable type: 'list'

Слово «нехэшируемый» в этом контексте означает, что этот «список» — это объект, который можно изменить.

Словари в Python полагаются на то, что каждый ключ имеет хеш-значение, определенный идентификатор для ключа. Если ключ может измениться, это хеш-значение не будет надежным. Таким образом, ключи всегда должны быть неизменяемыми, хешируемыми типами данных, такими как числа или строки.

**Пустой словарь**

В словаре ничего не должно быть. Иногда нам нужно создать пустой словарь, когда мы планируем заполнить его позже на основе других входных данных.

Мы можем создать пустой словарь следующим образом:

>>empty\_dict = {}

**Добавить ключ**

Чтобы добавить в словарь одну пару ключ: значение, мы можем использовать синтаксис:

dictionary[key] = value

Например, если бы у нас был словарь меню из первого упражнения:

>>menu = {"oatmeal": 3, "avocado toast": 6, "carrot juice": 5, "blueberry muffin": 2}

И мы хотели добавить новый товар, «чизкейк» за 8 долларов, мы могли бы использовать:

>>menu["cheesecake"] = 8

Теперь меню выглядит так:

>>{"oatmeal": 3, "avocado toast": 6, "carrot juice": 5, "blueberry muffin": 2, "cheesecake": 8}

**Задание**

1. Создайте пустой словарь с именем animals\_in\_zoo.
2. Гуляя по зоопарку, вы видите 8 зебр. Добавьте «зебры» в animals\_in\_zoo в качестве ключа со значением 8.
3. Добавьте «обезьяны» в animals\_in\_zoo в качестве ключа со значением 12.
4. Покидая зоопарк, вы опечалены тем, что не видели динозавров. Добавьте "динозавров" в animals\_in\_zoo в качестве ключа со значением 0.
5. Выведите в консоль animals\_in\_zoo.

**Добавить несколько ключей**

Если мы хотим добавить в словарь сразу несколько пар ключ: значение, мы можем использовать метод .update ().

Рассмотрим словарь с показаниями датчиков из предыдущего упражнения:

>>sensors = {"living room": 21, "kitchen": 23, "bedroom": 20}

Если бы мы хотели добавить 3 новые комнаты, мы могли бы использовать:

>>sensors.update({"pantry": 22, "guest room": 25, "patio": 34})

Это добавит все три элемента в словарь датчиков.

Теперь sensors выглядят так:

>>{"living room": 21, "kitchen": 23, "bedroom": 20, "pantry": 22, "guest room": 25, "patio": 34}

**Задание**

1. К словарю user\_ids, представленному ниже, добавьте двух новых пользователей:

* theLooper, с id of 138475
* stringQueen, с id of 85739

>>ser\_ids = {"teraCoder": 9018293, "proProgrammer": 119238}

1. Выведите в консоль результат

**Перезаписать значения**

Мы знаем, что можем добавить ключ, используя такой синтаксис:

>>menu["тост с авокадо "] = 7

Это создаст ключ «тост с авокадо» и установит значение, равное 7. Но что, если у нас уже есть запись «тост с авокадо» в словаре?

В этом случае наше присвоение значения перезапишет существующее значение, прикрепленное к ключу «тост с авокадо».

>>menu = {"oatmeal": 3, "avocado toast": 6, "carrot juice": 5, "blueberry muffin": 2}  
>>menu["oatmeal"] = 5  
>>print(menu)

Этоn код выдаст следующий результат:

{"oatmeal": 5, "avocado toast": 6, "carrot juice": 5, "blueberry muffin": 2}

Обратите внимание, что  "oatmeal" поменялось на  5.

**Задание**

* 1. Добавьте к словарю oscar\_winners  ключ «"Supporting Actress"» и установите значение «Viola Davis».

oscar\_winners = {"Best Picture": "La La Land", "Best Actor": "Casey Affleck", "Best Actress": "Emma Stone", "Animated Feature": "Zootopia"}

* 1. Не меняя определения словаря oscar\_winners, измените значение, связанное с ключом «Best Picture», на «Moonlight».

**Генерирование словаря**

Допустим, у нас есть два списка, которые мы хотим объединить в словарь, например список студентов и список их роста в дюймах:

>>names = ['Jenny', 'Alexus', 'Sam', 'Grace']  
>>heights = [61, 70, 67, 64]

Python позволяет создавать словарь, используя генерирование словаря, с таким синтаксисом:

>>students = {key:value for key, value in zip(names, heights)}  
>>#students is now {'Jenny': 61, 'Alexus': 70, 'Sam': 67, 'Grace': 64}

Помните, что zip () объединяет два списка в итеративный тип, содержащий кортежи с элементами списка, соединенными вместе. Это генерирование словаря:

* Берет пару из итератора кортежей
* Называет элементы в паре ключом (тот, который изначально был из списка имен) и значение (тот, который был первоначально из списка роста студентов)
* Создает элемент "ключ: значение" в словаре студентов.
* Повторяет шаги 1-3 для всего итератора пар

**Задание**

1. У вас есть два списка, в которых представлены некоторые напитки, продаваемые в кафе, и миллиграммы кофеина в каждом. Сначала создайте переменную с именем zipped\_drinks, которая является итеративным типом, включающим пары между списком напитков и списком кофеина.
2. Создайте словарь с именем drinks\_to\_caffeine, используя генерирование словаря, которое проходит через итератор zipped\_drinks и превращает каждую пару кортежей в элемент ключ: значение.

drinks = ["espresso", "chai", "decaf", "drip"]

caffeine = [64, 40, 0, 120]

**Заключение**

На данный момент мы узнали:

* Как создать словарь
* Как добавить элементы в словарь
* Как обновить элементы в словаре
* Как использовать понимание списка для создания словаря из двух списков

**Задание**

1. Мы создаем музыкальный стриминговый сервис. Мы предоставили два списка, в которых представлены песни в библиотеке пользователя и количество воспроизведений каждой песни.

Используя генерирование словаря, создайте словарь под названием plays, который проходит через zip (песни, playcounts) и создает пару song: playcount для каждой песни в песнях и каждого playcount в playcounts.

songs = ["Like a Rolling Stone", "Satisfaction", "Imagine", "What's Going On", "Respect", "Good Vibrations"]

playcounts = [78, 29, 44, 21, 89, 5]

1. Выведите plays.
2. После этого добавьте к plays новую запись. Запись должна быть для песни "Purple Haze", а количество воспроизведений - 1.
3. Этот пользователь заразился лихорадкой Ареты Франклин и прослушал «Respect» 5 раз.
4. Создайте словарь, называемый library, который имеет две пары ключ: значение:

«The Best Songs» со значением plays, (т.е. словарь, который вы создали использовать в качестве значения)

ключ "Sunday Feelings" со значением пустого словаря

1. Выведите library.

**Использование словарей**

Данное занятие посвящено следующим вопросам:

* Использование ключа, чтобы получить значение из словаря
* Проверка наличия ключей
* Перебор ключей и значений в словарях

**Получить значение по ключу**

Если у вас есть словарь, вы можете получить доступ к его значениям, указав ключ. Например, представим, что у нас есть словарь, в котором высота зданий отображается в метрах:

>>building\_heights = {"Burj Khalifa": 828, "Shanghai Tower": 632, "Abraj Al Bait": 601, "Ping An": 599, "Lotte World Tower": 554.5, "One World Trade": 541.3}

Затем мы можем получить доступ к данным в нем следующим образом:

>>> building\_heights["Burj Khalifa"]  
828

**Задание**

1. Мы предоставили словарь, в котором элементы стихий соотносятся со знаками зодиака. Выведите в консоль список знаков зодиака, связанных со стихией «земля».

zodiac\_elements = {"water": ["Cancer", "Scorpio", "Pisces"], "fire": ["Aries", "Leo", "Sagittarius"], "earth": ["Taurus", "Virgo", "Capricorn"], "air":["Gemini", "Libra", "Aquarius"]}

1. Выведите в консоль список знаков стихии огня.

**Недействительные ключи**

Допустим, у нас есть словарь высот зданий из последнего упражнения:

>>building\_heights = {"Burj Khalifa": 828, "Shanghai Tower": 632, "Abraj Al Bait": 601, "Ping An": 599, "Lotte World Tower": 554.5, "One World Trade": 541.3}

Что, если бы мы хотели узнать высоту Landmark 81 в Хошимине? Мы могли бы попробовать:

>>print(building\_heights["Landmark 81"])

Но «Landmark 81» не существует в качестве ключа в словаре building\_heights! Таким образом, это приведет к ошибке KeyError:

KeyError: 'Landmark 81'

Один из способов избежать этой ошибки - сначала проверить, существует ли ключ в словаре:

>>key\_to\_check = "Landmark 81"  
>>if key\_to\_check in building\_heights:  
>>  print(building\_heights["Landmark 81"])

Это не приведет к ошибке, потому что key\_to\_check в building\_heights вернет False, и поэтому мы никогда не пытаемся получить доступ к ключу.

**Задание**

1. Запустите код. Он должен выдать KeyError! «energy» не существует как один из элементов.
2. Добавьте ключ «energy» к zodiac\_elements. Он должен соответствовать значению «Not a Zodiac element». Запустите код. Устранено ли это KeyError?

zodiac\_elements = {"water": ["Cancer", "Scorpio", "Pisces"], "fire": ["Aries", "Leo", "Sagittarius"], "earth": ["Taurus", "Virgo", "Capricorn"], "air":["Gemini", "Libra", "Aquarius"]}

**Try/Except для получения ключа**

Мы увидели, что можем избежать KeyErrors, сначала проверив, есть ли ключ в словаре. Другой метод, который мы могли бы использовать, — это try / except:

key\_to\_check = "Landmark 81"  
try:  
  print(building\_heights[key\_to\_check])  
except KeyError:  
  print("That key doesn't exist!")

Когда мы пытаемся получить доступ к несуществующему ключу, программа перейдет в блок except и выведет «Этот ключ не существует!».

**Задание**

1. Используйте блок try/except чтобы попытаться вывести уровень кофеина в матче. Если возникла ошибка KeyError, выведите «Неизвестный уровень кофеина».

caffeine\_level = {"espresso": 64, "chai": 40, "decaf": 0, "drip": 120}

1. Над блоком try добавьте в словарь «matcha» со значением 30

**Безопасно получить ключ**

В последнем упражнении мы видели, что можно добавить в словарь пару ключ: значение, чтобы избежать KeyError. Это решение неустойчиво. Мы не можем предсказать каждый ключ, который может вызвать пользователь, и добавить все эти значения-заполнители в наш словарь!

В словарях есть метод .get () для поиска значения вместо my\_dict [key], которую мы использовали ранее. Если ключ, который вы пытаетесь использовать в методе .get (), не существует, по умолчанию он вернет None:

>>building\_heights = {"Burj Khalifa": 828, "Shanghai Tower": 632, "Abraj Al Bait": 601, "Ping An": 599, "Lotte World Tower": 554.5, "One World Trade": 541.3}  
>>#this line will return 632:  
>>building\_heights.get("Shanghai Tower")  
>>#this line will return None:  
>>building\_heights.get("My House")

Возможно указать значение, которое будет возвращено, если ключ не существует. Например, мы можем вернуть высоту здания, равную 0, если желаемое здание отсутствует в словаре:

>>> building\_heights.get('Shanghai Tower', 0)  
632  
>>> building\_heights.get('Mt Olympus', 0)  
0  
>>> building\_heights.get('Kilimanjaro', 'No Value')  
'No Value'

**Задание**

1. Используйте .get (), чтобы получить значение идентификатора пользователя teraCoder, со значением по умолчанию 100000, если пользователь не существует. Сохраните его в переменной tc\_id. Выведите tc\_id в консоль.
2. Используйте .get (), чтобы получить значение идентификатора пользователя superStackSmash, со значением по умолчанию 100000, если пользователь не существует. Сохраните его в переменной stack\_id. Вывести stack\_id в консоль.

user\_ids = {"teraCoder": 100019, "pythonGuy": 182921, "samTheJavaMaam": 123112, "lyleLoop": 102931, "keysmithKeith": 129384}

**Удалить ключ**

Иногда мы хотим получить ключ и удалить его из словаря. Представьте, что мы проводим розыгрыш, и у нас есть словарь, сопоставляющий номера билетов с призами:

>>raffle = {223842: "Teddy Bear", 872921: "Concert Tickets", 320291: "Gift Basket", 412123: "Necklace", 298787: "Pasta Maker"}

Когда мы получаем номер билета, мы хотим вернуть приз, а также удалить эту пару из словаря, поскольку приз уже был отдан. Для этого мы можем использовать метод .pop (). Как и в случае с .get (), мы можем предоставить значение по умолчанию, если ключ не существует в словаре:

>>> raffle.pop(320291, "No Prize")  
"Gift Basket"  
>>> raffle  
{223842: "Teddy Bear", 872921: "Concert Tickets", 412123: "Necklace", 298787: "Pasta Maker"}  
>>> raffle.pop(100000, "No Prize")  
"No Prize"  
>>> raffle  
{223842: "Teddy Bear", 872921: "Concert Tickets", 412123: "Necklace", 298787: "Pasta Maker"}  
>>> raffle.pop(872921, "No Prize")  
"Concert Tickets"  
>>> raffle  
{223842: "Teddy Bear", 412123: "Necklace", 298787: "Pasta Maker"}

**Получить все ключи**

Иногда нам нужно оперировать всеми ключами в словаре. Например, если у нас есть словарь учащихся математического класса и их оценок:

>>test\_scores = {"Grace":[80, 72, 90], "Jeffrey":[88, 68, 81], "Sylvia":[80, 82, 84], "Pedro":[98, 96, 95], "Martin":[78, 80, 78], "Dina":[64, 60, 75]}

Мы хотим получить список учащихся в классе без учета их оценок. Мы можем сделать это с помощью встроенной функции list ():

>>> list(test\_scores)  
["Grace", "Jeffrey", "Sylvia", "Pedro", "Martin", "Dina"]

В словарях также есть метод .keys (), который возвращает объект dict\_keys. Объект dict\_keys — это объект представления, который обеспечивает просмотр текущего состояния словаря без возможности изменения пользователем чего-либо. Объект dict\_keys, возвращаемый функцией .keys (), представляет собой набор ключей в словаре. Вы не можете добавлять или удалять элементы из объекта dict\_keys, но его можно использовать вместо списка для итерации:

>>for student in test\_scores.keys():  
>>   print(student)

выведет:

Grace  
Jeffrey  
Sylvia  
Pedro  
Martin  
Dina

**Задание**

1. Создайте переменную с именем users и назначьте ей объект dict\_keys для всех ключей словаря user\_ids.

user\_ids = {"teraCoder": 100019, "pythonGuy": 182921, "samTheJavaMaam": 123112, "lyleLoop": 102931, "keysmithKeith": 129384}

num\_exercises = {"functions": 10, "syntax": 13, "control flow": 15, "loops": 22, "lists": 19, "classes": 18, "dictionaries": 18}

1. Создайте переменную с именем classes и назначьте ее объектом dict\_keys для всех ключей словаря num\_exercises.
2. Вывести users в консоль.
3. Вывести lessons на консоли.

**Получить все значения**

В словарях есть метод .values (), который возвращает объект dict\_values (точно так же, как объект dict\_keys, но для значений!) Со всеми значениями в словаре. Его можно использовать вместо списка для итерации:

>>test\_scores = {"Grace":[80, 72, 90], "Jeffrey":[88, 68, 81], "Sylvia":[80, 82, 84], "Pedro":[98, 96, 95], "Martin":[78, 80, 78], "Dina":[64, 60, 75]}  
  
>>for score\_list in test\_scores.values():  
>>   print(score\_list)

Выдаст:

[80, 72, 90]  
[88, 68, 81]  
[80, 82, 84]  
[98, 96, 95]  
[78, 80, 78]  
[64, 60, 75]

Нет встроенной функции для получения всех значений в виде списка, но если вы действительно хотите, вы можете использовать:

>>list(test\_scores.values())

**Задание**

1. Создайте переменную с именем total\_exercises и установите ее равной 0.
2. Просмотрите значения в списке num\_exercises и добавьте каждое значение в переменную total\_exercises.
3. Выведите в консоль переменную total\_exercises.

>>num\_exercises = {"functions": 10, "syntax": 13, "control flow": 15, "loops": 22, "lists": 19, "classes": 18, "dictionaries": 18}

**Получить все ключи и значения**

Вы можете получить как ключи, так и значения с помощью метода .items (). Подобно .keys () и .values (), он возвращает объект dict\_list. Каждый элемент dict\_list, возвращаемый функцией .items (), представляет собой кортеж, состоящий из:

(key, value)

поэтому для итерации вы можете использовать этот синтаксис:

>>biggest\_brands = {"Apple": 184, "Google": 141.7, "Microsoft": 80, "Coca-Cola": 69.7, "Amazon": 64.8}  
  
>>for company, value in biggest\_brands.items():  
>>   print(company + " has a value of " + str(value) + " billion dollars. ")

В результате будет выведено::

Apple has a value of 184 billion dollars.  
Google has a value of 141.7 billion dollars.  
Microsoft has a value of 80 billion dollars.  
Coca-Cola has a value of 69.7 billion dollars.  
Amazon has a value of 64.8 billion dollars.

**Заключение**

В этом уроке вы узнали, как просматривать словари и получать доступ к ключам и значениям разными способами. В частности, вы узнали, как:

* Использовать ключ, чтобы получить значение из словаря
* Проверить наличие ключей
* Удалить пару ключ: значение из словаря
* Перебирать ключи и значения в словарях

**Задание**

1. Мы предоставили колоду карт Таро. Вы собираетесь вытащить из колоды три карты своего прошлого, настоящего и будущего.

tarot = { 1:  "The Magician", 2:  "The High Priestess", 3:  "The Empress", 4: "The Emperor", 5: "The Hierophant", 6:  "The Lovers", 7:  "The Chariot", 8: "Strength", 9:  "The Hermit", 10: "Wheel of Fortune", 11: "Justice", 12:  "The Hanged Man", 13: "Death", 14:  "Temperance", 15: "The Devil", 16:  "The Tower", 17:  "The Star", 18: "The Moon", 19: "The Sun", 20:  "Judgement", 21:  "The World", 22: "The Fool"}

Создайте пустой словарь под названием spread.

1. Первая карта, которую вы берете, — это карта 13. Извлеките значение, присвоенное ключу 13, из словаря Таро и назначьте его как значение к ключу «прошлое».
2. Вторая карта, которую вы берете, — это карта 22. Извлеките значение, присвоенное ключу 22, из словаря Таро и назначьте его как значение ключу «настоящеее».
3. Третья карта, которую вы берете, — это карта 10. Извлеките значение, присвоенное ключу 10, из словаря Таро и назначьте его как значение к ключу «будущее»

Файлы

**Чтение файла**

Компьютеры используют файловые системы для хранения и извлечения данных. Каждый файл представляет собой отдельный контейнер со связанной информацией. Если вы когда-либо сохраняли документ, загружали песню или даже отправляли электронное письмо, вы создали файл где-то на каком-то компьютере. Даже script.py, программа Python, которую вы редактируете представляет собой файл.

Итак, как мы взаимодействуем с файлами с помощью Python? Мы собираемся научиться читать и записывать различные типы файлов с помощью кода. Допустим, у нас есть файл с именем real\_cool\_document.txt со следующим содержимым:

real\_cool\_document.txt

Wowsers!

Мы могли бы прочитать этот файл так:

with open('real\_cool\_document.txt') as cool\_doc:

cool\_contents = cool\_doc.read()

print(cool\_contents)

Это открывает файловый объект с именем cool\_doc и создает новый блок, в котором вы можете прочитать содержимое открытого файла. Затем мы читаем содержимое файла cool\_doc с помощью cool\_doc.read () и сохраняем полученную строку в переменной cool\_contents. Затем мы выводим cool\_contents, который выводит инструкцию Wowsers !.

**Итерации по строкам**

Когда мы читаем файл, мы можем захотеть получить весь документ в одной строке, например, как при методе .read (). Но что, если бы мы хотели сохранить каждую строку в переменной? Мы можем использовать функцию .readlines () для чтения текстового файла построчно, вместо того, чтобы считывать сразу целиком. Допустим, у нас есть файл:

keats\_sonnet.txt

To one who has been long in city pent,

’Tis very sweet to look into the fair

And open face of heaven,—to breathe a prayer

Full in the smile of the blue firmament.

script.py

>>with open('keats\_sonnet.txt') as keats\_sonnet:

>> for line in keats\_sonnet.readlines():

>> print(line)

Приведенный выше сценарий создает временный файловый объект keats\_sonnet, который указывает на файл keats\_sonnet.txt. Затем он перебирает каждую строку в документе и выводит в консоль весь файл.

**Чтение строки**

Иногда вам не нужно перебирать весь файл. Для этого есть другой файловый метод .readline (), который будет читать только одну строку за раз. Если весь документ будет прочитан таким образом построчно, последующие вызовы .readline () не вызовут ошибки, но начнут возвращать пустую строку (""). Предположим, у нас есть такой файл:

millay\_sonnet.txt

I shall forget you presently, my dear,

So make the most of this, your little day,

Your little month, your little half a year,

Ere I forget, or die, or move away,

script.py

>>with open('millay\_sonnet.txt') as sonnet\_doc:

>> first\_line = sonnet\_doc.readline()

>> second\_line = sonnet\_doc.readline()

>>print(second\_line)

Этот сценарий также создает файловый объект с именем sonnet\_doc, который указывает на файл millay\_sonnet.txt. Затем он считывает первую строку с помощью sonnet\_doc.readline () и сохраняет ее в переменной first\_line. Затем он сохраняет вторую строку (так что извлеките максимум из этого, вашего маленького дня) в переменную second\_line и затем выводит ее.

**Запись файла**

Чтение файла — это хорошо, но что, если мы хотим создать собственный файл? С Python мы можем это сделать. Оказывается, нашей функции open (), которую мы используем для открытия файла для чтения, нужен еще один аргумент, чтобы открыть файл для записи.

script.py

>>with open('generated\_file.txt', 'w') as gen\_file:

>> gen\_file.write("What an incredible file!")

Здесь мы передаем аргумент 'w' функции open (), чтобы указать на открытие файла в режиме записи. Аргументом по умолчанию является 'r', а передача 'r' в open () открывает файл в режиме чтения, как мы это делали.

Этот код создает новый файл в той же папке, что и script.py, и дает ему текст «Какой невероятный файл!». Важно отметить, что если уже существует файл с именем generated \_file.txt, он полностью перезапишет этот файл, удалив все его содержимое, которое было раньше.

**Добавление в файл**

Так что, возможно, вас беспокоит полное удаление и перезапись существующих файлов. Разве нельзя просто добавить строку в файл, не удаляя его полностью? Конечно, есть! Вместо того, чтобы открывать файл с помощью аргумента «w» для режима записи, мы открываем его с помощью «a» для режима добавления. Если у нас есть сгенерированный файл со следующим содержимым:

generated\_file.txt

This was a popular file...

Затем мы можем добавить в этот файл еще одну строку со следующим кодом:

script.py

>>with open('generated\_file.txt', 'a') as gen\_file:

>> gen\_file.write("... and it still is")

В приведенном выше коде мы открываем файловый объект во временной переменной gen\_file. Эта переменная указывает на файл created\_file.txt и, поскольку он открыт в режиме добавления, добавляет строку ... и по-прежнему является новой строкой в файле. Если бы вы открыли файл после запуска скрипта, он бы выглядел так:

generated\_file.txt

This was a popular file...

... and it still is

s

Обратите внимание, что открытие файла в режиме добавления с 'a' в качестве аргумента функции open () означает, что использование метода .write () файлового объекта добавляет все, что передается в конец файла, в новой строке. Если бы мы снова запустили script.py, это выглядело бы так: created\_file.txt:

generated\_file.txt

This was a popular file...

... and it still is

... and it still is

Обратите внимание, что мы второй раз добавили к файлу "... и это по-прежнему"! Это потому, что в script.py мы открыли файл created\_file.txt в режиме добавления.

**"with"**

До сих пор мы открывали эти файлы с помощью this with block, но кажется немного странным, что мы можем использовать нашу файловую переменную только в блоке с отступом. Это почему? Ключевое слово with вызывает так называемый диспетчер контекста для файла, для которого мы вызываем open (). Этот диспетчер контекста заботится об открытии файла, когда мы вызываем open (), а затем закрытии файла после того, как мы выйдем из блока с отступом.

Почему так сложно закрыть файл? Что ж, большинство других аспектов нашего кода связаны с тем, что контролирует сам Python. Все создаваемые вами переменные: целые числа, списки, словари - все это объекты Python, и Python знает, как их очистить, когда с ними будет работать. Поскольку ваши файлы существуют вне вашего скрипта Python, нам нужно сообщить Python, когда мы закончили с ними, чтобы он мог закрыть соединение с этим файлом. Если оставить соединение с файлом открытым без необходимости, это может повлиять на производительность или повлиять на другие программы на вашем компьютере, которые могут пытаться получить доступ к этому файлу.

Синтаксис with заменяет старые способы доступа к файлам, когда вам нужно вручную вызывать .close () для объекта файла. Мы все еще можем открыть файл и добавить к нему старый синтаксис, если не забыть закрыть соединение с файлом после этого.

fun\_cities\_file = open('fun\_cities.txt', 'a')

# We can now append a line to "fun\_cities".

fun\_cities\_file.write("Montréal")

# But we need to remember to close the file

fun\_cities\_file.close()

В приведенном выше сценарии мы добавили «Montréal» в качестве новой строки в нашем файле fun\_cities.txt. Однако, поскольку мы использовали синтаксис старого стиля, нам пришлось не забыть закрыть файл после этого. Поскольку это обязательно более подробный (требуется как минимум еще одна строка кода), но не более выразительный, использование with является предпочтительным.

**Что такое файл CSV?**

Текстовые файлы - не единственное, что может читать Python, но это единственное, что нам не нужна дополнительная библиотека синтаксического анализа для понимания. Файлы CSV — это пример текстового файла, который накладывает структуру на свои данные. CSV означает значения, разделенные запятыми, а файлы CSV обычно используются для экспорта данных из программного обеспечения для работы с электронными таблицами (например, Microsoft Excel или Google Sheets) в переносимый формат. Таблица, которая выглядит следующим образом:

|  |  |  |
| --- | --- | --- |
| Name | Username | Email |
| Roger Smith | rsmith | wigginsryan@yahoo.com |
| Michelle Beck | mlbeck | hcosta@hotmail.com |
| Ashley Barker | a\_bark\_x | a\_bark\_x@turner.com |
| Lynn Gonzales | goodmanjames | lynniegonz@hotmail.com |

В файле CSV те же самые данные будут отображаться следующим образом:

users.csv

Name,Username,Email

Roger Smith,rsmith,wigginsryan@yahoo.com

Michelle Beck,mlbeck,hcosta@hotmail.com

Ashley Barker,a\_bark\_x,a\_bark\_x@turner.com

Lynn Gonzales,goodmanjames,lynniegonz@hotmail.com

Обратите внимание, что первая строка файла CSV на самом деле не представляет никаких данных, а только метки данных, которые присутствуют в остальной части файла. Остальные строки файла такие же, как и в программе для работы с электронными таблицами только вместо того, чтобы быть разделенными на разные ячейки, они разделены ... ну, я полагаю, будет справедливо сказать, что они разделены запятыми.

**Чтение файла CSV**

users.csv

Name,Username,Email

Roger Smith,rsmith,wigginsryan@yahoo.com

Michelle Beck,mlbeck,hcosta@hotmail.com

Ashley Barker,a\_bark\_x,a\_bark\_x@turner.com

Lynn Gonzales,goodmanjames,lynniegonz@hotmail.com

Несмотря на то, что мы можем без проблем читать эти строки как текст, есть способы получить доступ к данным в формате, более подходящем для целей программирования. В Python мы можем преобразовать эти данные в словарь с помощью объекта DictReader библиотеки CSV. Вот как мы создадим список адресов электронной почты всех пользователей в приведенной выше таблице:

>>import csv

>>list\_of\_email\_addresses = []

>>with open('users.csv', newline='') as users\_csv:

>> user\_reader = csv.DictReader(users\_csv)

>> for row in user\_reader:

>> list\_of\_email\_addresses.append(row['Email'])

В приведенном выше коде мы сначала импортируем нашу библиотеку csv, которая дает нам инструменты для анализа нашего файла CSV. Затем мы создаем пустой список list\_of\_email\_addresses, который позже заполним адресами электронной почты из нашего CSV. Затем открываем файл users.csv с временной переменной users\_csv.

Мы передаем дополнительный аргумент ключевого слова newline = '' в функцию открытия файла open (), чтобы случайно не принять разрыв строки в одном из наших полей данных за новую строку в нашем CSV (подробнее об этом читайте в Python документация).

После открытия нашего нового файла CSV мы используем csv.DictReader (users\_csv), который преобразует строки нашего файла CSV в словари Python, для которых мы можем использовать методы доступа. Ключи словаря по умолчанию - это записи в первой строке нашего CSV-файла. Поскольку первая строка нашего CSV вызывает третье поле в нашем CSV «Электронная почта», мы можем использовать это как ключ в каждой строке нашего DictReader.

Когда мы перебираем строки нашего объекта user\_reader, мы получаем доступ ко всем строкам в нашем CSV как к словарям (за исключением первой строки, которую мы использовали для маркировки ключей нашего словаря). Получив доступ к ключу «Электронная почта» в каждой из этих строк, мы можем получить адрес электронной почты в этой строке и добавить его в наш list\_of\_email\_addresses.

**Чтение различных типов файлов CSV**

Мы действовали так, как будто файлы CSV — это файлы со значениями, разделенными запятыми. Это правда, что CSV означает это, но также верно и то, что в наши дни допустимы другие способы разделения значений.

Раньше люди называли файлы значений, разделенных табуляцией, файлами TSV, но по мере роста популярности других разделителей все осознали, что создание нового формата файла. [A-z] sv для каждого используемого символа-разделителя не является устойчивым.

Поэтому мы вызываем все файлы со списком различных значений в CSV-файл, а затем используем разные разделители (например, запятую или табуляцию), чтобы указать, где начинаются и заканчиваются различные значения (разделение значений).

Допустим, у нас была адресная книга. Поскольку в адресах обычно используются запятые, нам потребуется использовать другой разделитель для нашей информации. Поскольку ни в одной из наших данных нет точек с запятой (;), мы можем их использовать.

addresses.csv

Name;Address;Telephone

Donna Smith;126 Orr Corner Suite 857\nEast Michael, LA 54411;906-918-6560

Aaron Osborn;6965 Miller Station Suite 485\nNorth Michelle, KS 64364;815.039.3661x42816

Jennifer Barnett;8749 Alicia Vista Apt. 288\nLake Victoriaberg, TN 51094;397-796-4842x451

Joshua Bryan;20116 Stephanie Stravenue\nWhitneytown, IA 87358;(380)074-6173

Andrea Jones;558 Melissa Keys Apt. 588\nNorth Teresahaven, WA 63411;+57(8)7795396386

Victor Williams;725 Gloria Views Suite 628\nEast Scott, IN 38095;768.708.3411x954

Обратите внимание на символ \ n, это escape-последовательность для новой строки. Возможность экранирования новой строки в наших данных символом \ n является причиной того, что мы передаем аргумент ключевого слова newline = '' функции open ().

Также обратите внимание, что во многих из этих адресов есть запятые! Ничего страшного, мы все равно сможем это прочитать. Если бы мы хотели, скажем, распечатать все адреса в этом CSV-файле, мы могли бы сделать следующее:

import csv

>>with open('addresses.csv', newline='') as addresses\_csv:

>> address\_reader = csv.DictReader(addresses\_csv, delimiter=';')

>> for row in address\_reader:

>> print(row['Address'])

Обратите внимание, что когда мы вызываем csv.DictReader, мы передаем параметр разделителя, который представляет собой строку, которая используется для выделения отдельных полей в CSV. Затем мы просматриваем CSV и распечатываем каждый из адресов.

**Запись файла CSV**

Естественно, если у нас есть возможность читать различные файлы CSV, хотелось бы иметь возможность записи этих файлов для дальнейшего использования полученных данных. Допустим, у нас есть большой список данных, которые мы хотим сохранить в файл CSV. Мы могли сделать следующее:

big\_list = [{'name': 'Fredrick Stein', 'userid': 6712359021, 'is\_admin': False}, {'name': 'Wiltmore Denis', 'userid': 2525942, 'is\_admin': False}, {'name': 'Greely Plonk', 'userid': 15890235, 'is\_admin': False}, {'name': 'Dendris Stulo', 'userid': 572189563, 'is\_admin': True}]

>>import csv

>>with open('output.csv', 'w') as output\_csv:

>> fields = ['name', 'userid', 'is\_admin']

>> output\_writer = csv.DictWriter(output\_csv, fieldnames=fields)

>> output\_writer.writeheader()

>> for item in big\_list:

>> output\_writer.writerow(item)

В нашем коде выше у нас был набор словарей с одинаковыми ключами для каждого, главный кандидат в CSV. Мы импортируем библиотеку csv, а затем открываем новый файл CSV в режиме записи, передав аргумент w функции open ().

Затем мы определяем поля, которые собираемся использовать, в переменную с именем fields. Затем мы создаем экземпляр нашего объекта записи CSV и передаем два аргумента. Первый - output\_csv, объект-обработчик файлов. Второй — это наш список полей полей, которые мы передаем в имена полей параметров ключевого слова.

Теперь, когда мы создали экземпляр нашего модуля записи CSV-файла, мы можем начать добавлять строки в сам файл! Сначала нам нужны заголовки, поэтому мы вызываем функцию writeheader () для объекта записи. Это записывает все названия полей, переданные в fieldnames, как первую строку в нашем файле. Затем мы перебираем наш big\_list данных. Каждый элемент в big\_list представляет собой словарь с каждым полем в полях в качестве ключей. Мы вызываем output\_writer.writerow () со словарями элементов, которые записывают каждую строку в файл CSV.

**Чтение файла JSON**

CSV - не единственный формат файлов, для которого Python имеет встроенную библиотеку. Мы также можем использовать файловые инструменты Python для чтения и записи JSON. JSON, сокращение от JavaScript Object Notation, представляет собой формат файла, вдохновленный языком программирования JavaScript.

Формат JSON поразительно похож на синтаксис словаря Python, поэтому файлы JSON могут быть легко читаемыми с точки зрения разработчика Python. Тем не менее, Python поставляется с пакетом json, который поможет нам преобразовать файлы JSON в реальные словари Python. Предположим, у нас есть файл JSON, подобный следующему:

purchase\_14781239.json

{

“user”: “ellen\_greg”,

“action”: “purchase”,

“item\_id”: “14781239”,

}

Мы могли бы прочитать это как словарь Python со следующим кодом:

json\_reader.py

>>import json

>>with open('purchase\_14781239.json') as purchase\_json:

>> purchase\_data = json.load(purchase\_json)

>>print(purchase\_data['user'])

Сначала мы импортируем пакет json. Мы открыли файл с помощью команды open (). Поскольку мы открываем его в режиме чтения, нам просто нужно передать имя файла. Сохраняем файл во временной переменной Purchase\_json.

Мы продолжаем синтаксический анализ Purchase\_json с помощью json.load (), создавая словарь Python из файла. Сохранение результатов в файле Purchase\_data означает, что мы можем с ним взаимодействовать. Мы распечатываем одно из значений файла JSON, введя объект Purchase\_data.

**Запись файла JSON**

Естественно, мы также можем использовать библиотеку json для перевода объектов Python в JSON. Это особенно полезно в тех случаях, когда вы используете библиотеку Python для обслуживания веб-страниц, вы также сможете обслуживать JSON. Допустим, у нас есть словарь Python, который мы хотели сохранить в виде файла JSON:

turn\_to\_json = {

'eventId': 674189,

'dateTime': '2015-02-12T09:23:17.511Z',

'chocolate': 'Semi-sweet Dark',

'isTomatoAFruit': True

}

Мы сможем создать файл JSON с этой информацией, выполнив следующие действия:

>>import json

>>with open('output.json', 'w') as json\_file:

>> json.dump(turn\_to\_json, json\_file)

Мы импортируем модуль json, открываем файл режима записи под переменной json\_file, а затем используем метод json.dump () для записи в файл. json.dump () принимает два аргумента: сначала объект данных, затем объект файла, который вы хотите сохранить.

Итоговое задание

1. Создайте файл csv, содержащий следующие сведения:

|  |  |  |  |
| --- | --- | --- | --- |
| ФИО | Должность | Дата найма | Оклад |
| Иванов Иван Иванович | Менеджер | 22.10.2013 | 250 000 |
| Сорокина Екатерина Матвеевна | Аналитик | 12.03.2020 | 75 000 |
| Струков Иван Сергеевич | Старший программист | 23.04.2012 | 150 000 |
| Корнеева Анна Игоревна | Ведущий программист | 22.02.2015 | 120 000 |
| Старчиков Сергей Анатольевич | Младший программист | 12.11.2021 | 50 000 |
| Бутенко Артем Андреевич | Архитектор | 12.02.2010 | 200 000 |
| Савченко Алина Сергеевна | Старший аналитик | 13.04.2016 | 100 000 |

1. Начальство решило выплачивать премии ко дню программиста в сентябре (13 сентября – день программиста). Премия будет составлять 3% от оклада каждому программисту. Реализуйте функцию, которая будет рассчитывать премию.
2. Начальство решило выплачивать премии к 8 марта всем сотрудницам, а к 23 февраля сотрудникам, равные 2000. Реализуйте функцию, которая будет рассчитывать премию.
3. Было решено провести индексацию зарплат сотрудников. Сотрудникам, которые отработали в компании более 10 лет, индексация будет равна 7% оклада, остальным – 5%.
4. Необходимо составить график отпусков сотрудников. Для этого необходимо в отдельный список занести тех сотрудников, которые отработали в компании более 6 месяцев. Реализовать функцию.
5. Реализуйте функции записи в csv и json.

**Для продвинутых:**

Выполнить все те же задания, только создайте классы Employee, который будет включать в себя свойства Фамилия, имя, отчество, должность, дата найма, оклад. Реализуйте инкапсуляцию. Не забудьте про геттеры и сеттеры. Внутри класса реализуйте методы из задания выше.

Создайте функцию, которая рассчитывает фонд оплаты труда.

Создайте функцию, осуществляющую построение столбчатой диаграммы с размером окладов по должностям.

Создайте функцию, которая рассчитывает налоговые отчисления (подоходный налог, фонд социального страхования).