**Назовите и дайте краткую характеристику основных классов языков программирования:**

**Процедурные** (**императивные**) языки — Процедурные языки программирования - используют процедуры (подпрограммы, методы или функции). содержат последовательность шагов для выполнения. В ходе выполнения программы любая процедура может быть вызвана из любой точки. При процедурном программировании программа разбивается на части в соответствии с алгоритмом. Характерным для процедурного языка является наличие оператора присваивания. Программа, написанная на императивном языке, очень похожа на приказы, выражаемые повелительным наклонением в естественных языках, то есть это последовательность команд, которые должен выполнить компьютер. Характеризуются последовательным, пошаговым изменением состояния вычислителя. При этом управление изменениями полностью определено и полностью контролируемо. Одна из характерных черт императивного программирования - наличие переменных с операцией "разрушающего присвоения".

**Непроцедурные** (**декларативные**) языки — это языки, при использовании которых в программе в явном виде указывается, какими свойствами должен обладать результат, но не говорится, каким способом он должен быть получен. Типичным примером таких языков являются языки логического программирования. Характерной особенностью декларативных языков является их декларативная семантика. Основная концепция декларативной семантики заключается в том, что смысл каждого оператора не зависит от того, как этот оператор используется в программе. Декларативная семантика намного проще семантики императивных языков, что может рассматриваться как преимущество декларативных языков над императивными. Способ решения задачи описывается при помощи зависимости функций друг от друга без указания последовательности шагов.

Функциональное программирование, как и другие модели "неимперативного" программирования, обычно применяется для решения задач, которые трудно сформулировать в терминах последовательных операций. Это, например, задачи распознавания образов, общение с пользователем на естественном языке, реализация экспертных систем, автоматизированное доказательство теорем, символьные вычисления.

В логическом программировании программа представляет из себя некоторую теорию (описанную на достаточно ограниченном языке), и утверждение, которое нужно доказать. В доказательстве этого утверждения и будет заключаться исполнение программы. Логическое программирование оказывается удобным для реализации сложных задач.

**Типизация в языках программирования (статическая и динамическая, явная и неявная, сильная и слабая). Реализация типизации в Python.**

Статическая – переменная, параметр подпрограммы, возвращаемое значение функции связывается с типом в момент объявления и тип не может быть изменен в процессе выполнения программы.

Динамическая – переменная связывается с типом в момент присваивания значения, тогда в разных участках программы одна и та же переменная может принимать разные типы.

Явная / неявная типизация. Явно-типизированные языки отличаются тем, что тип новых переменных / функций / их аргументов нужно задавать явно. Соответственно языки с неявной типизацией перекладывают эту задачу на компилятор / интерпретатор.

Сильная / слабая типизация (также иногда говорят строгая / нестрогая). Сильная типизация выделяется тем, что язык не позволяет смешивать в выражениях различные типы и не выполняет автоматические неявные преобразования, например, нельзя вычесть из строки множество. Языки со слабой типизацией выполняют множество неявных преобразований автоматически, даже если может произойти потеря точности или преобразование неоднозначно.

В Python динамическая сильная неявная типизация

Операторы сравнения и логические операторы. Инструкция if...else.

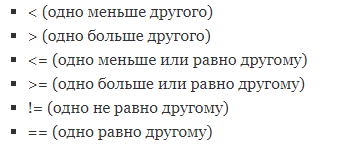
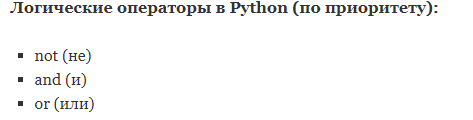
Обозначаем сравнения словами "равно", "больше", "меньше". В языках программирования используются специальные знаки, подобные тем, которые используются в математике: > (больше), < (меньше), >= (больше или равно), <= (меньше или равно), == (равно), != (не равно).

Логические выражения типа kByte >= 1023 являются простыми, так как в них выполняется только одна логическая операция

Существуют более сложные специальные операторы, объединяющие два и более простых логических выражения. Широко используются два оператора – так называемые логические И (**and**) и ИЛИ (**or**).

Чтобы получить True при использовании оператора **and**, необходимо, чтобы результаты обоих простых выражений, которые связывает данный оператор, были истинными. Если хотя бы в одном случае результатом будет False, то и все сложное выражение будет ложным.

Чтобы получить True при использовании оператора **or**, необходимо, чтобы результат хотя бы одного простого выражения, входящего в состав сложного, был истинным. В случае оператора **or** сложное выражение становится ложным лишь тогда, когда ложны оба составляющие его простые выражения.

В языке Python есть еще унарный логический оператор **not**, т. е. отрицание. Он превращает правду в ложь, а ложь в правду. Унарный он потому, что применяется к одному выражению, стоящему после него, а не справа и слева от него как в случае бинарных and и or.

Операторы сравнения, логические операторы, if…else - все это связано с понятием

ветвящихся (нелинейных) алгоритмов.

В отличии от линейных, в них возможны разные действия, зависящие от данных на

каком-либо этапе выполнения программы.

Оператор условного перехода if используется как раз для перехода по

определенной ветки в зависимости от данных.

Итак, условная инструкция в Питоне имеет следующий синтаксис:

if *Условие*:

*Блок инструкций 1*

else:

*Блок инструкций 2*

Блок инструкций 1 будет выполнен, если Условие истинно. Если Условие ложно, будет выполнен Блок инструкций 2.

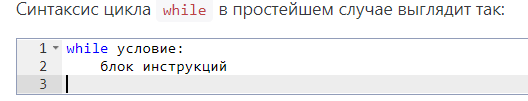
В условной инструкции может отсутствовать слово else и последующий блок. Такая инструкция называется неполным ветвлением.

Для выделения блока инструкций, относящихся к инструкции if или else в языке Питон используются отступы. Все инструкции, которые относятся к одному блоку, должны иметь равную величину отступа, то есть одинаковое число пробелов в начале строки. Рекомендуется использовать отступ в 4 пробела и не рекомедуется использовать в качестве отступа символ табуляции.

Циклы в Python, работа и устройство цикла while. Типичное применение while.

While:

Цикл while (“пока”) позволяет выполнить одну и ту же последовательность действий, пока проверяемое условие истинно. Условие записывается до тела цикла и проверяется до выполнения тела цикла. Как правило, цикл while используется, когда невозможно определить точное значение количества проходов исполнения цикла.



При выполнении цикла while сначала проверяется условие. Если оно ложно, то выполнение цикла прекращается и управление передается на следующую инструкцию после тела цикла while. Если условие истинно, то выполняется инструкция, после чего условие проверяется снова и снова выполняется инструкция. Так продолжается до тех пор, пока условие будет истинно. Как только условие станет ложно, работа цикла завершится и управление передастся следующей инструкции после цикла.

После тела цикла можно написать слово else: и после него блок операций, который будет выполнен *один раз* после окончания цикла, когда проверяемое условие станет неверно

Если во время выполнения Питон встречает инструкцию break внутри цикла, то он сразу же прекращает выполнение этого цикла и выходит из него. При этом ветка else исполняться не будет. Разумеется, инструкцию break осмысленно вызывать только внутри инструкции if, то есть она должна выполняться только при выполнении какого-то особенного условия.

Другая инструкция управления циклом — continue (продолжение цикла). Если эта инструкция встречается где-то посередине цикла, то пропускаются все оставшиеся инструкции до конца цикла, и исполнение цикла продолжается со следующей итерации.

Сравнение:

Цикл while по сравнению с for является достаточно медленным.

У циклов for/while по синтаксису также есть else. Else проверяет, был ли произведен выход из цикла инструкцией break, или же "естественным" образом. Блок инструкций внутри else выполнится только в том случае, если выход из цикла произошел без помощи break.

Циклы в Python, работа и устройство цикла for, типичное применение range и enumerate в цикле for.

**For:**

В цикле for указывается переменная и множество значений, по которому будет пробегать переменная. Множество значений может быть задано списком, кортежем, строкой или диапазоном. Мы используем **цикл** в тех случаях, когда вам нужно повторить что-нибудь n-ное количество раз



### **Функция range**

Как правило, циклы for используются либо для повторения какой-либо последовательности действий заданное число раз, либо для изменения значения переменной в цикле от некоторого начального значения до некоторого конечного.

Для повторения цикла некоторое заданное число раз n можно использовать цикл for вместе с функцией range:

В качестве n может использоваться числовая константа, переменная или произвольное арифметическое выражение. Если значение n равно нулю или отрицательное, то тело цикла не выполнится ни разу.

Встроенная в Python функция enumerate() применяется для итерируемых

коллекций (строки, списки, словари и др.) и создает объект, который

генерирует кортежи, состоящие из двух элементов - индекса элемента и

самого элемента.

Выход из цикла можно осуществить при помощи инструкции break

Также можно пропустить текущую итерацию при помощи инструкции continue

У циклах for/while по синтаксису также есть else. Else проверяет, был ли произведен

выход из цикла инструкцией break, или же "естественным" образом. Блок

инструкций внутри else выполнится только в том случае, если выход из цикла

произошел без помощи break.

Списки в Python. Обращение к элементам списка и создание срезов. Обход списка и поиск элементов в списке.

Для хранения таких данных можно использовать структуру данных, называемую в Питоне список. Список представляет собой последовательность элементов, пронумерованных от 0, как символы в строке. Список можно задать перечислением элементов списка в квадратных скобках, например, список можно задать так.



Или просто объявив его:

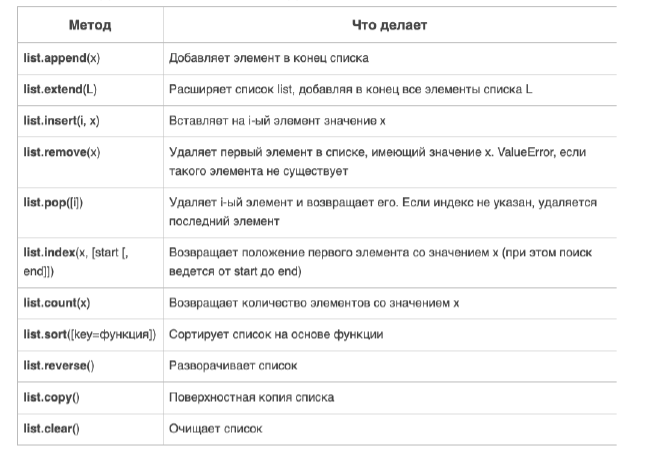
x = []

Или вызывав конструктор списка

x = list()

Cписок может содержать любое количество любых объектов (в том

числе и вложенные списки, tuple, dict и т. д), или не содержать ничего.



В Python, кроме индексов, существуют ещё и срезы.

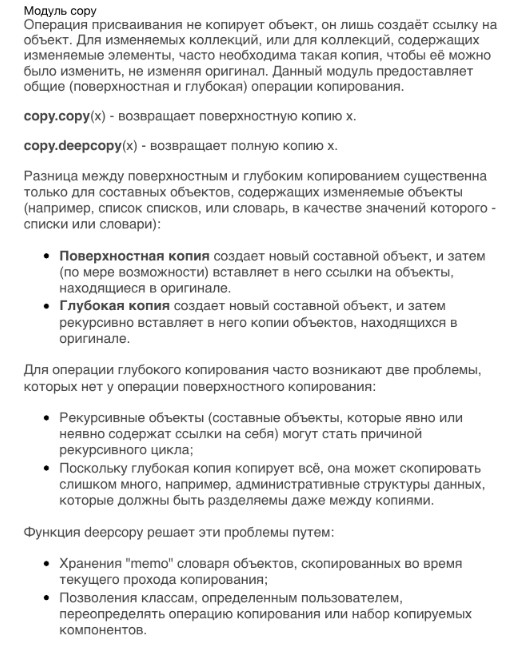
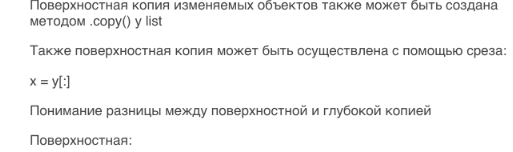
Срезы используются не только в списках, но и в большинстве остальных Коллекциях.

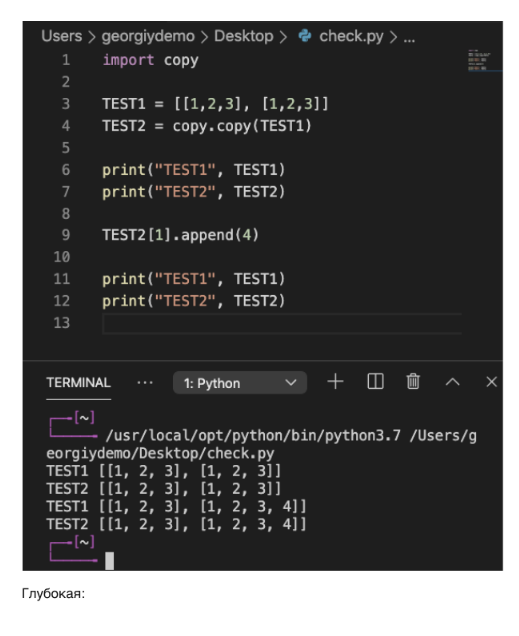
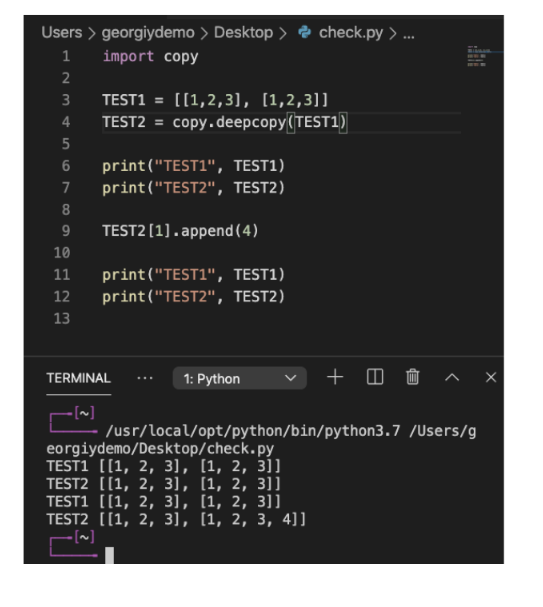
Списки в Python. Ключевые операции, проводящие к

изменению списка и порождающие измененные

списки.

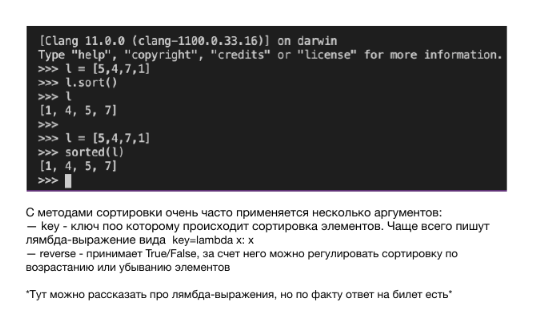
См. билет 7.

Копирование списков.



## 10. Выражения генераторы и генераторы списков в Python. Использование условий в генераторах.

## Агрегирующие функции sum(), min(), max(). Сортировка



Строки. Операции над строками. Основные методы строк.

## -Упорядоченные последовательности символов, используемые для хранения и представления текстовой инфы, поэтому с помощью строк можно работать со всем, что может быть представлено в текстовой форме. Строка может объявляться: “ или ‘

Узнать длину строки можно при помощи функции len. Любой другой объект в Питоне можно перевести к строке, которая ему соответствует. Для этого нужно вызвать функцию str(), передав ей в качестве параметра объект, переводимый в строку.

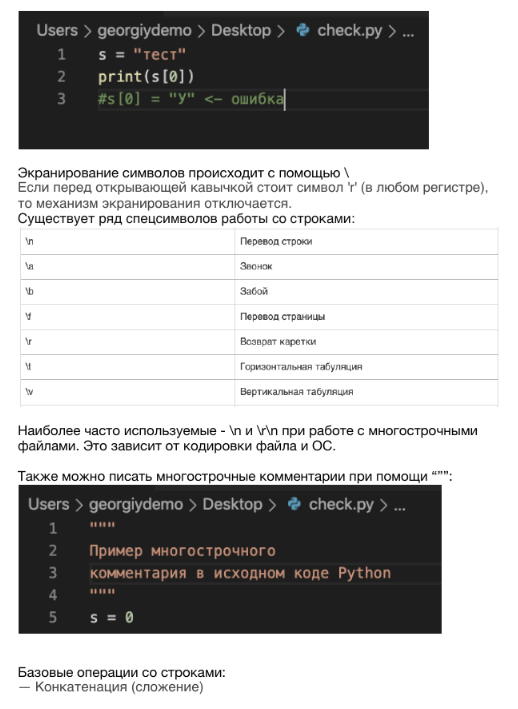
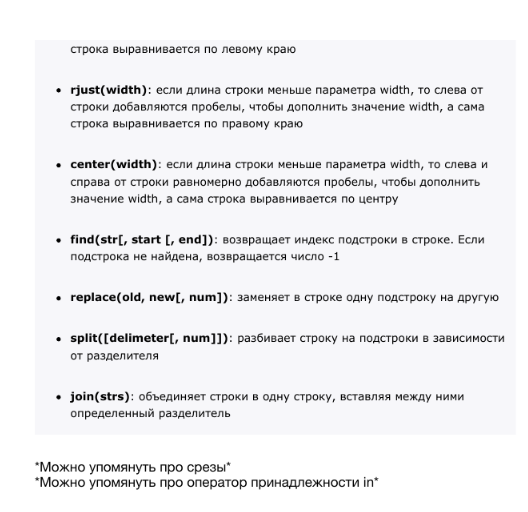
Срез (slice) — извлечение из данной строки одного символа или некоторого фрагмента подстроки или подпоследовательности. Есть три формы срезов. Самая простая форма среза: взятие одного символа строки, а именно, S[i] — это срез, состоящий из одного символа, который имеет номер i. При этом считается, что нумерация начинается с числа 0. То есть если S = 'Hello', то S[0] == 'H', S[1] == 'e', S[2] == 'l', S[3] == 'l', S[4] == 'o'. Номера символов в строке (а также в других структурах данных: списках, кортежах) называются индексом. Срез с двумя параметрами: S[a:b] возвращает подстроку из b - a символов, начиная с символа c индексом a, то есть до символа с индексом b, не включая его.

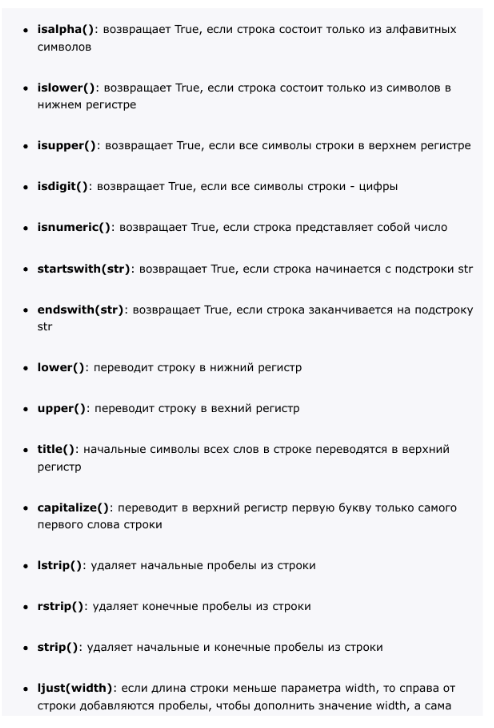
Метод — это функция, применяемая к объекту, в данном случае — к строке. Метод вызывается в виде Имя\_объекта.Имя\_метода(параметры). Например, S.find("e") — это применение к строке S метода find с одним параметром "e". Метод find находит в данной строке (к которой применяется метод) данную подстроку (которая передается в качестве параметра). Функция возвращает индекс первого вхождения искомой подстроки. Если же подстрока не найдена, то метод возвращает значение -1.

Аналогично, метод rfind возвращает индекс последнего вхождения данной строки (“поиск справа”).

Метод replace заменяет все вхождения одной строки на другую. Формат: S.replace(old, new) — заменить в строке S все вхождения подстроки old на подстроку new.

Метод count подсчитывает количество вхождений одной строки в другую строку. Простейшая форма вызова S.count(T) возвращает число вхождений строки T внутри строки S. При этом подсчитываются только непересекающиеся вхождения.

При указании трех параметров S.count(T, a, b), будет выполнен подсчет числа вхождений строки T в срезе S[a:b].



## 

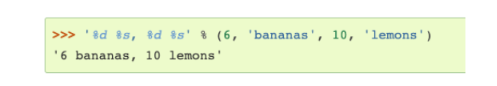
## Форматирование текста в Python.

Если для подстановки требуется только один аргумент, то значение - сам аргумент:



А если несколько, то значениями будут являться все аргументы со строками подстановки (обычных или именованных):

## 



## Преобразование между строками и списками в Python

## 

## Словари в Python. Итерирование по словарям, преобразование между словарями и списками в Python. Операции с представлениями словарей

## Обычные списки (массивы) представляют собой набор пронумерованных элементов, то есть для обращения к какому-либо элементу списка необходимо указать его номер. Номер элемента в списке однозначно идентифицирует сам элемент. Но идентифицировать данные по числовым номерам не всегда оказывается удобно. Словари являются по своей сути коллекциями без индексирования (т.е. у элементов нет своего порядкового номера). каждый элемент словаря состоит из двух объектов: *ключа* и *значения*. Ключ идентифицирует элемент словаря, значение является данными, которые соответствуют данному ключу. Значения ключей — уникальны, двух одинаковых ключей в словаре быть не может. В языке Питон ключом может быть произвольный неизменяемый тип данных: целые и действительные числа, строки, кортежи. Ключом в словаре не может быть множество, но может быть элемент типа frozenset: специальный тип данных, являющийся аналогом типа set, который нельзя изменять после создания. Значением элемента словаря может быть любой тип данных, в том числе и изменяемый.

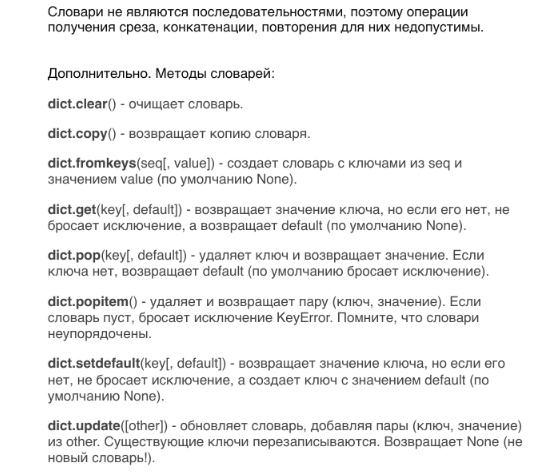
## Когда нужно использовать словари

Словари нужно использовать в следующих случаях:

* Подсчет числа каких-то объектов. В этом случае нужно завести словарь, в котором ключами являются объекты, а значениями — их количество.
* Хранение каких-либо данных, связанных с объектом. Ключи — объекты, значения — связанные с ними данные. Например, если нужно по названию месяца определить его порядковый номер, то это можно сделать при помощи словаря Num['January'] = 1; Num['February'] = 2; ....
* Установка соответствия между объектами (например, “родитель—потомок”). Ключ — объект, значение — соответствующий ему объект.
* Если нужен обычный массив, но масимальное значение индекса элемента очень велико, и при этом будут использоваться не все возможные индексы (так называемый “разреженный массив”), то можно использовать ассоциативный массив для экономии памяти.

Пустой словарь можно создать при помощи функции dict () или пустой пары фигурных скобок {} (вот почему фигурные скобки нельзя использовать для создания пустого множества).

Основная операция: получение значения элемента по ключу, записывается так же, как и для списков: A[key]. Если элемента с заданным ключом нет в словаре, то возникает исключение KeyError. Другой способ определения значения по ключу — метод get: A.get(key). Если элемента с ключом get нет в словаре, то возвращается значение None. В форме записи с двумя аргументами A.get(key, val) метод возвращает значение val, если элемент с ключом key отсутствует в словаре. Проверить принадлежность элемента словарю можно операциями in и not in, как и для множеств. Для добавления нового элемента в словарь нужно просто присвоить ему какое-то значение: A[key] = value. Для удаления элемента из словаря можно использовать операцию del A[key] (операция возбуждает исключение KeyError, если такого ключа в словаре нет. Вот два безопасных способа удаления элемента из словаря. Быстро проверить, есть ли значение val среди всех значений элементов словаря A можно так: val in A.values(), а организовать цикл так, чтобы в переменной key был ключ элемента, а в переменной val, было его значение можно так:



## 

## Операции со словарями, учитывающие возможное отсутствие ключа. Операции многоэлементного изменения словарей. Операции поэлементного извлечения из словаря и их использование.

## Множества в Python. Основные способы создания, получения и изменения значений. Обход множеств.

Множество в языке Питон — это структура данных, эквивалентная множествам в математике. Множество может состоять из различных элементов, порядок элементов в множестве неопределен. В множество можно добавлять и удалять элементы, можно перебирать элементы множества, можно выполнять операции над множествами (объединение, пересечение, разность). Можно проверять принадлежность элемента множеству.

## Задание множеств:

Множество задается перечислением всех его элементов в фигурных скобках. Исключением является пустое множество, которое можно создать при помощи функции set(). Если функции set передать в качестве параметра список, строку или кортеж, то она вернёт множество, составленное из элементов списка, строки, кортежа. Например:



Каждый элемент может входить в множество только один раз, порядок задания элементов неважен.

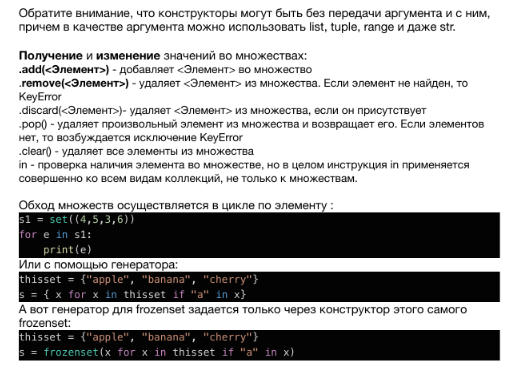
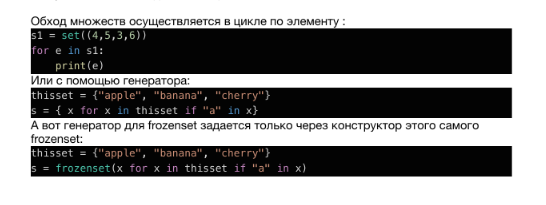
## Работа с элементами множеств. Узнать число элементов в множестве можно при помощи функции len.

Перебрать все элементы множества (в неопределенном порядке!) можно при помощи цикла for:

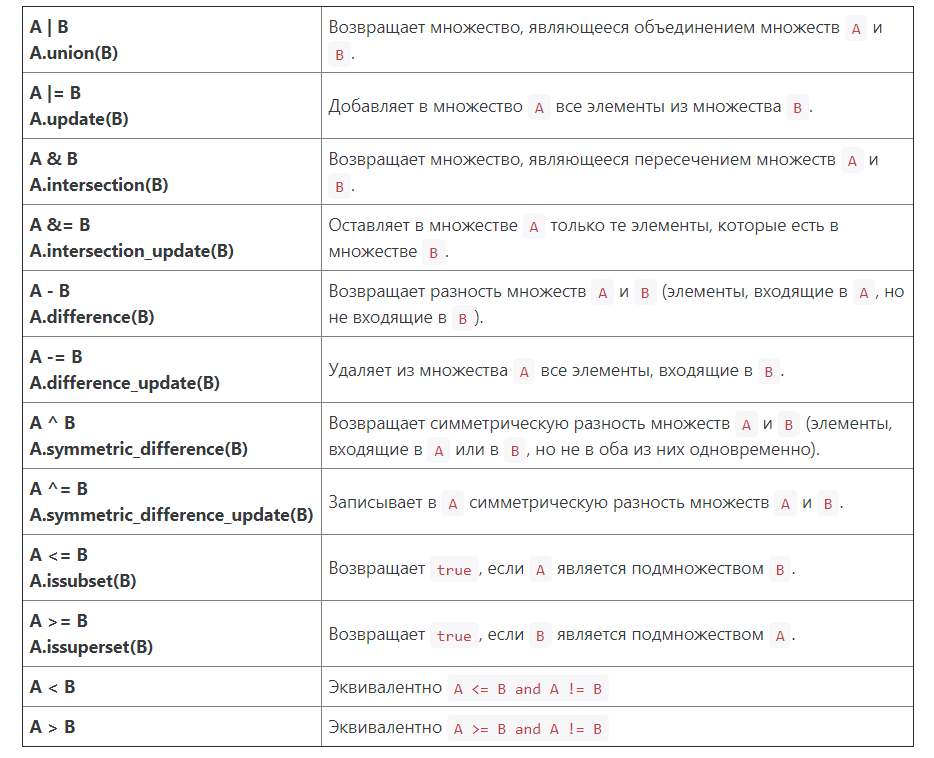
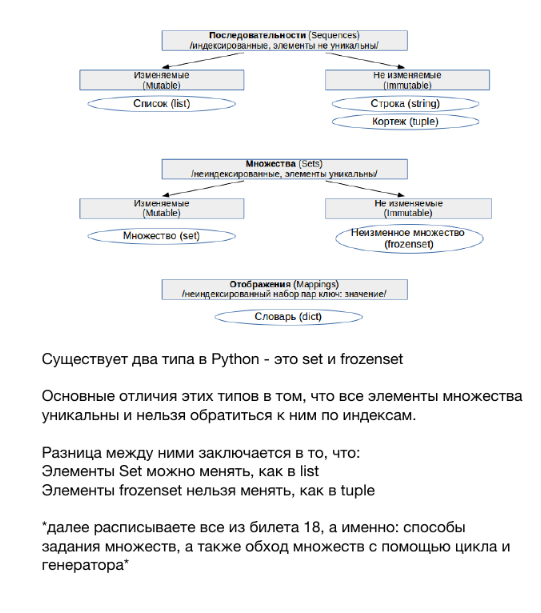
Проверить, принадлежит ли элемент множеству можно при помощи операции in, возвращающей значение типа bool. Аналогично есть противоположная операция not in. Для добавления элемента в множество есть метод add:

Для удаления элемента x из множества есть два метода: discard и remove. Их поведение различается только в случае, когда удаляемый элемент отсутствует в множестве. В этом случае метод discard не делает ничего, а метод remove генерирует исключение KeyError.

Наконец, метод pop удаляет из множества один случайный элемент и возвращает его значение. Если же множество пусто, то генерируется исключение KeyError.

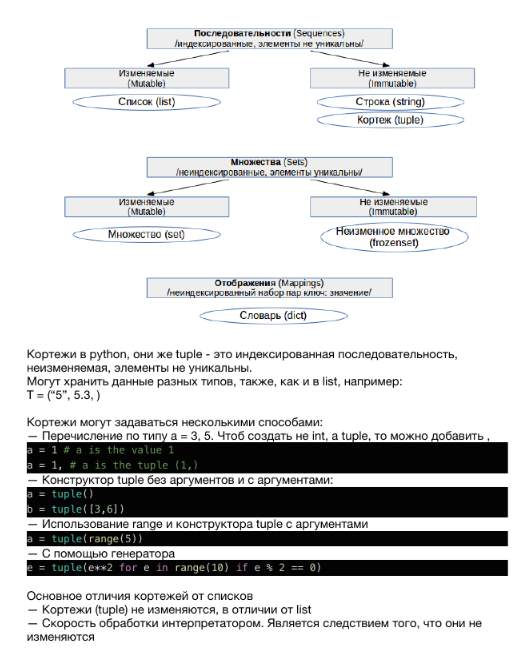
Из множества можно сделать список при помощи функции list.

## Выполнение основных операций с парой множеств в Python.



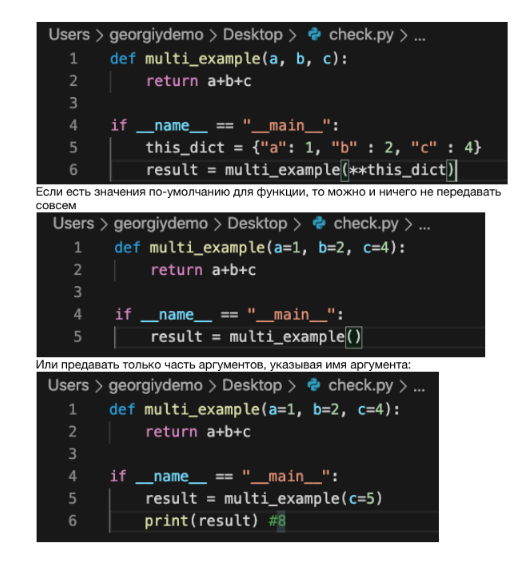
Операции с множествами:

## Кортежи в Python. Отличия кортежей от списков. Распаковка и частичная распаковка кортежей.

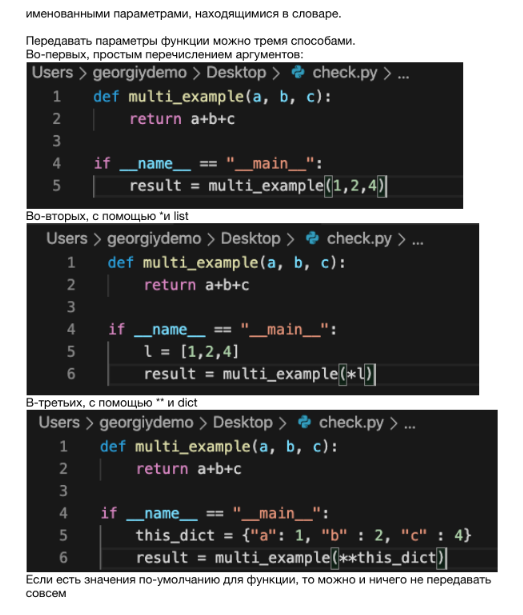


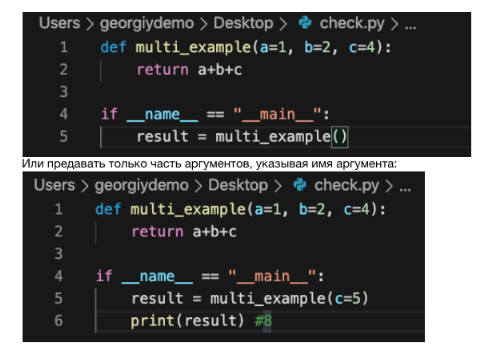
## Объявление и вызов функции в Python. Параметры функции со значением по умолчанию и комментирование функции. Получение информации о функции. Способы передачи параметров при вызове функции.

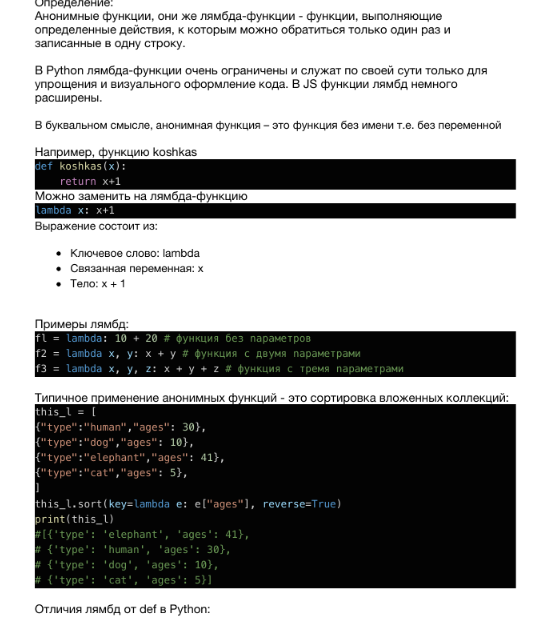
## 

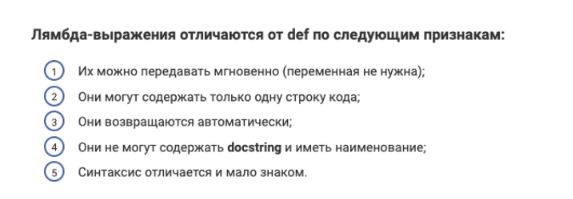


## Передача переменного количества параметров (именованных и не именованных) в функции Python. Вызов функции с позиционными параметрами, находящимися в списке, и именованными параметрами, находящимися в словаре.

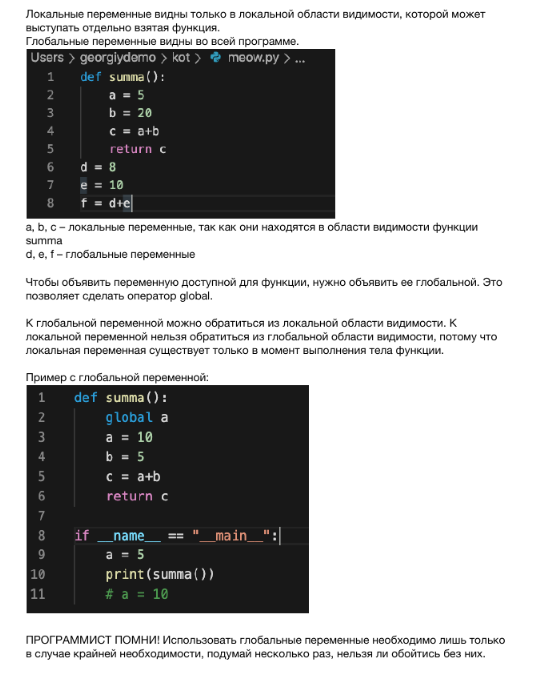




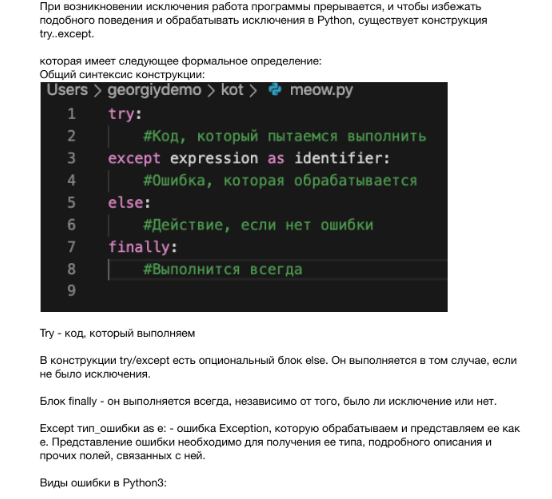
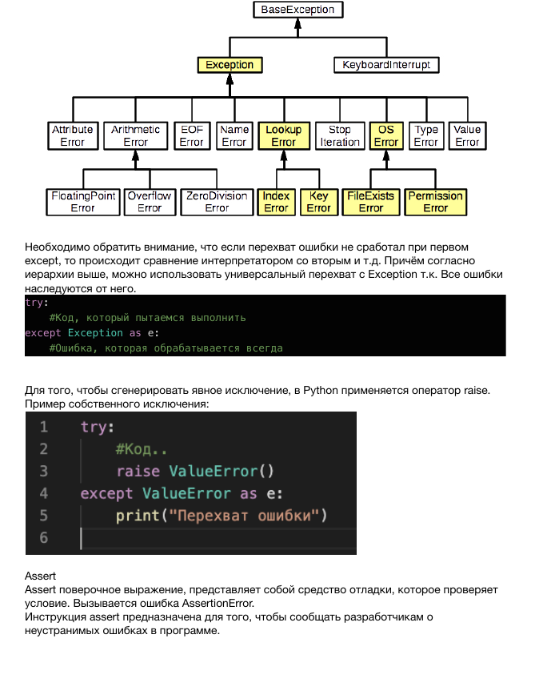
Анонимные функции. Типичное применение анонимных функций.



## Глобальные и локальные переменные.



## Обработка исключительных ситуаций в Python. Создание пользовательских исключений и инструкция assert.



## Базовые операции для работы с файлами в Python. Использование инструкции with … as для работы с файлами.

## 

Модули в Python. Варианты синтаксиса импорта модуля и объектов модуля. Применение импортированных объектов. Порядок поиска модулей и специфика их загрузки.

