

МАТЕРИАЛЫ ДЛЯ ПОДГОТОВКИ К ЗАЧЕТУ.

1 БЛОК.

1. Программирование. Что такое языки программирования? Когда появился первый язык и как он назывался? Что такое среда разработки (IDE)? Что такое код/программа? Что такое компиляция программы? Понятие алгоритма. В чем плюсы и минусы Python?
2. Компиляция программы. Ввод/Вывод данных. Примеры.
3. Переменные. Типы данных. Операции над переменными. Примеры.
4. Переменные. Тип str. Строки. Методы работы со строками. Примеры кода на каждый метод - минимум 5 примеров.
5. Переменные. Тип str. Строки. В чем отличие int, float, str? Понятие F-строки. Примеры.
6. Переменные и типы данных. Комментарии в Python, Docstring, Примеры.
7. Переменные. Типы данных. Функция type(). Преобразование типов (int(), ..). Преобразование типов (list(), dict(), tuple(), set()). Примеры.
8. Условная конструкция IF-ELSE. Зачем нужен ELIF? С примерами.
9. Циклы. Виды циклов в Python. Схема работы цикла while. Примеры.
10. Циклы. Виды циклов в Python. Схема работы цикла for. Функция range(). Примеры.
11. Циклы. Схемы циклов. Понятие бесконечного цикла. Примеры.
12. Три важных оператора (break, continue, return). Циклы while и for. Чем цикличность отличается от не-цикличности? Примеры.
13. Файловая система. Работа с файлами. Режимы работы с файлами.
14. Программирование. Что такое компиляция программы? Понятие алгоритма. PEP. Типы PEP.

2 БЛОК.

1. Коллекции в Python. Списки. Срезы. Примеры.
2. Коллекции в Python. Списки. Матрица элементов. Типы матриц. Примеры.
3. Коллекции в Python. Списки. Матрица элементов. Матричные алгоритмы. Примеры.
4. Коллекции в Python. Списки, кортежи. Чем матрица от обычного списка отличается? Сфера применения матриц. Определитель матрицы. Примеры.
5. Коллекции в Python. Списки, кортежи. Отличия. Примеры.
6. Коллекции в Python. Списки, срезы. Типы циклов пробежки по списку. Примеры.
7. Коллекции в Python. Списки, множества. Отличия. Примеры.
8. Коллекции в Python. Списки, словари. Отличия. Примеры.

1 БЛОК.

1. (идеально расписанный 1 вопрос)

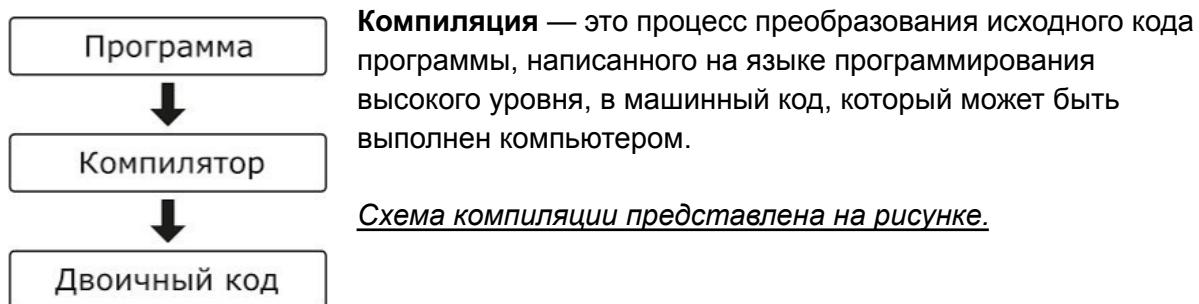
Программирование. Что такое языки программирования? Когда появился первый язык и как он назывался? Что такое среда разработки (IDE)? Что такое код/программа? Что такое компиляция программы? Понятие алгоритма. В чем плюсы и минусы Python?

Программирование — это процесс создания программного обеспечения путем написания инструкций для компьютеров на одном или нескольких языках программирования. Программирование включает в себя анализ проблемы, разработку алгоритма, написание кода, его тестирование и отладку.

Языки программирования — это формальные языки, которые разработаны для написания программ, то есть наборов инструкций, которые выполняются компьютером. Примеры языков: Java, Python, C++, JavaScript, TypeScript... Первый язык программирования был создан в 1950-х годах. **Форран (Fortran)**, от компании IBM.

Интегрированная среда разработки (IDE) — это программное обеспечение, которое предоставляет разработчикам набор инструментов для написания, редактирования, компиляции, отладки и управления кодом. IDE обычно включает редактор кода, компилятор или интерпретатор, отладчик и инструменты для управления версиями. Примеры популярных IDE включают Microsoft Visual Studio, PyCharm, Eclipse и IntelliJ IDEA.

Код — это текст инструкций, написанных на языке программирования, который определяет, что компьютер должен делать. **Программа** — это законченный набор этих инструкций, который может быть выполнен для достижения конкретной задачи.



Что такое интерпретатор??

Интерпретатор — это программа, которая выполняет код непосредственно, строка за строкой, преобразуя его в машинные инструкции в реальном времени. В отличие от компилятора, интерпретатор не создает отдельного исполняемого файла. Вместо этого он анализирует и выполняет исходный код каждый раз при запуске программы.

Этапы интерпретатора:



- **Лексический анализ (Лексер):** На этом этапе интерпретатор читает исходный код и разбивает его на токены. Токены — это минимальные единицы смысла в коде, такие как ключевые слова, идентификаторы, операторы, числа и символы. Лексический анализ помогает преобразовать исходный код в структуру, которую легче обрабатывать на следующих этапах.
- **Синтаксический анализ (Парсер):** Парсер принимает токены, созданные лексером, и строит на их основе синтаксическое дерево (дерево разбора). Это дерево отображает структурные отношения между токенами и определяет, как токены комбинируются для формирования выражений и инструкций.
- **Генерация промежуточного представления (IR):** После синтаксического анализа интерпретатор может создать промежуточное представление (IR) кода. IR — это абстрактное представление программы, которое облегчает выполнение и оптимизацию. IR может быть в форме абстрактного синтаксического дерева (AST) или трехадресного кода.
- **Выполнение (Интерпретация):** На этом этапе интерпретатор выполняет промежуточное представление кода, интерпретируя инструкции по одной и выполняя соответствующие операции. Интерпретатор может использовать стек вызовов и таблицы символов для отслеживания переменных и функций.

Алгоритм — это конечный, четко определенный набор инструкций для выполнения определенной задачи или решения проблемы. Алгоритмы являются основой программирования и могут быть представлены в виде псевдокода, блок-схем или непосредственно на языке программирования. Они должны быть точными, последовательными и завершенными, чтобы обеспечивать правильное выполнение задач.

Python — это высокоуровневый язык программирования, известный своей простотой и читаемостью. Он широко используется в веб-разработке, научных вычислениях, искусственном интеллекте, анализе данных и автоматизации.

Плюсы:

- простота и читаемость кода;
- много разных модулей и библиотек;
- активное сообщество программистов;
- поддержка разных концептов программирования (ООП, функциональное и тп)

Минусы:

- низкая скорость выполнения;
- автоматическое управление памятью (что накладывает ограничения);
- ограниченная многопоточность;
- необходимость использования интерпретатора;

2. (расписанный 2 вопрос)

Компиляция программы. Ввод/Вывод данных. Примеры.



Компиляция программы — это процесс преобразования исходного кода, написанного на языке программирования высокого уровня, в машинный код, который может быть непосредственно выполнен компьютером. Этот процесс включает несколько этапов, которые позволяют преобразовать понятный человеку код в понятный машине. Компиляция обычно происходит один раз, после чего полученный исполняемый файл может быть запущен многократно без необходимости повторной компиляции.

Двоичный/Машинный код — это код состоящий из нулей (0) и единиц (1).

(можно еще сказать про интерпретатор и его этапа, также отличия от компилятора)

Ввод/Вывод (I/O) — это процессы получения данных от пользователя или другой программы и передачи данных пользователю или другой программе. Ввод/вывод являются основными функциями взаимодействия программы с внешним миром.

input() — это встроенная функция в Python, предназначенная для ввода данных с клавиатуры. Она позволяет пользователю программы ввести информацию, которая затем может быть использована внутри программы. Когда функция `input()` вызывается, программа приостанавливается и ждет, пока пользователь введет данные и нажмет клавишу Enter.

Как работает `input()`

- Ожидание ввода:** Когда `input()` вызывается, программа останавливается и ожидает, пока пользователь введет текст.
- Чтение ввода:** Пользователь вводит текст с клавиатуры.
- Возвращение строки:** После нажатия клавиши Enter введенный текст возвращается как строка (тип `str`).

print() — это встроенная функция в Python, используемая для вывода данных на экран. Она позволяет выводить строки, числа и другие объекты в стандартный поток вывода, которым обычно является консоль или терминал.

Как работает `print()`

- Прием аргументов:** `print()` принимает один или несколько аргументов, которые нужно вывести на экран.
- Форматирование вывода:** Аргументы преобразуются в строки и объединяются с пробелами между ними.
- Вывод:** Полученная строка выводится в стандартный поток вывода.

Пример с `input()`:

Ввод целого числа с клавиатуры в переменную `x`.

```
x = int(input("Введите число: "))
```

Ввод вещественного числа (с плав.точкой).

```
float_x = float(input("Введите вещественное число: "))
```

Ввод строки (уже можно преобразование типа данных не делать!).

```
text = input("Введите текст: ")
```

Функция `INPUT` всегда возвращает строковый тип данных (`str`) именно поэтому мы и не делаем преобразование типа данных когда хотим ввести просто строку или текст.

Пример `print()`:

Простой вывод сообщение (текст или строка):

```
print("This is my test text!")
```

Простой вывод значения переменной (без доп текста):

```
a = 5
```

```
print(a)
```

Простой вывод значения переменной уже с текстом для красоты:

```
print("value is equal", a)
```

или так

```
print("value is equal " + str(a))
```

Пример с переносом строки (`\n`)

```
print("This is my test text!\n", "Hello world\n", "Read file\n")
```

3. (расписанный 3 вопрос)

Переменные. Типы данных. Операции над переменными. Примеры.

Переменные — это именованные (значит имеют имя) области памяти, используемые для хранения данных. Они позволяют программам сохранять и манипулировать данными. То есть **Переменная** - это хранилище данных и значений.

Главные параметры которые должна иметь переменная при ее создании:

- имя переменной
- значение переменной

Основные правила именования переменных:

1. Имя переменной может содержать только латинские буквы, числа и символ нижнего подчеркивания
2. Имя переменной не должно содержать пробелов
3. Имя переменной не должно начинаться с цифры
4. Регистр важен: var и Var это разные переменные
5. В Python можно задавать переменные на русском языке (но так делать не рекомендуется - лучше задавать на английском языке)

Синтаксис создания переменной:

```
имя переменной =  
значение
```

Пример создания переменной:

```
name = "Alice"  
age = 30
```

Тип данных - это тип значения переменной.

Основные типы данных:

- int - целые числа; (размер 4 Байта)
- float - вещественные числа;
- str - строка;
- bool - логический тип данных;

Дополнительные типы данных:

- list - списки;
- tuple - кортеж;
- set - множество;
- dict - словарь;
- complex - комплексное число;
- None - пустой тип;

Операторы присваивания: =, +=, -=, *=, /= и другие;

= это простое присваивание;

+= и др это уже сокращенное; (y += 1 это тоже самое, что и y = y + 1)

Множественное присваивание — это удобная конструкция, которая позволяет присвоить значения нескольким переменным одновременно в одной строке кода. Например так: `a, b, c = 1, 2, 3`

Операции над переменными:

1) Арифметические операции:

- Сложение: `+`
- Вычитание: `-`
- Умножение: `*`
- Деление: `/`
- Целочисленное деление: `//`
- Остаток от деления: `%`
- Возвведение в степень: `**`

2) Операции над строками:

- Операция `+` это объединение строк;
- Операция `*` это повторение строки;
- `[]` используются для доступа по индексу;
- можно использовать срезы;

3) Сравнительные операции:

Сравнительные операции (равно, неравно, больше, меньше и др):

```
x = 5
y = 10

# Операции сравнения
print(x == y) # False, равно
print(x != y) # True, не равно
print(x > y) # False, больше
print(x < y) # True, меньше
print(x >= y) # False, больше или равно
print(x <= y) # True, меньше или равно
```

`True` - истина (верно)

`False` - ложь (значит неверно)

4) Логические операции:

- Логическое НЕ (`not`) - инвертирует (меняет на противоположное) значение булевой переменной.
- Логическое ИЛИ (`or`) - истина, если один элемент условия будет истинным.
- Логическое И (`and`) - истина, если оба элемента условия истинны.

Истина определяется как `True` (1), Ложь определяется как `False`(0)

Существует специальная таблица истинности, которая содержит в себе различные необходимые операции: инверсия, конъюнкция и тп;

A	B	Отрицание Инверсия (НЕ) $\neg A$	Конъюнкция Логическое умножение (И) $A \wedge B$	Дизъюнкция Логическое сложение (ИЛИ) $A \vee B$
0	0	1	0	0
0	1	1	0	1
1	0	0	0	1
1	1	0	1	1

(по операциям еще желательно накинуть примеры)

4. (расписанный 4 вопрос)

Переменные. Тип str. Строки. Методы работы со строками. Примеры кода на каждый метод - минимум 5 примеров.

(часть из 3-го вопроса)

Тип данных str — это один из основных типов данных в Python, который используется для хранения текстовой информации. Строки представляют собой последовательности символов и заключаются в одиночные, двойные или тройные кавычки. Строки в Python обладают множеством методов для обработки и манипуляции текстом.

Методы:

- **str.lower():** Преобразует все символы строки в нижний регистр.
- **str.upper():** Преобразует все символы строки в верхний регистр.
- **str.replace(old, new):** Заменяет в строке все вхождения подстроки old на new.
- **str.split(separator):** Разбивает строку на список подстрок по указанному разделителю.
- **str.strip():** Удаляет пробелы в начале и в конце строки.
- **str.find(substring):** Возвращает индекс первого вхождения подстроки substring. Если подстрока не найдена, возвращает -1.
- **str.join(iterable):** Объединяет элементы итерации в одну строку, используя текущую строку в качестве разделителя.
- **str.startswith(prefix):** Возвращает True, если строка начинается с указанного префикса.
- **str.endswith(suffix):** Возвращает True, если строка заканчивается указанным суффиксом.
- и другие методы;

Эти методы делают работу со строками в Python простой и эффективной.
функция len() - позволяет определить длину указанной строки;

Примеры кода:

```
s = "Hello, World!"  
print(len(s)) # 13
```

```
s = "Hello, World!"  
print(s.lower()) # "hello, world!"
```

и другие примеры...

5. (расписанный 5 вопрос)

Переменные. Тип str. Строки. В чем отличие int, float, str? Понятие F-строки. Примеры.

(часть из 3го и 4го вопроса)

int (целые числа):

- Представляет целочисленные значения, например, -10, 0, 42.
- В Python нет ограничения на размер целого числа, кроме доступной памяти.
- Можно выполнять арифметические операции, такие как сложение, вычитание, умножение и деление.

float (числа с плавающей запятой):

- Представляет дробные числа, например, 3.14, -0.001, 2.0.
- Хранит числа в формате с плавающей запятой с ограниченной точностью.
- Используется для представления дробных результатов арифметических операций.

str (строки):

- Представляет текстовые данные, заключенные в одинарные, двойные или тройные кавычки.
- Может содержать любые символы, включая буквы, цифры и специальные символы.
- Поддерживает множество операций и методов для работы с текстом.

F-строки — это специальный синтаксис в Python, который позволяет встраивать значения переменных и выражений непосредственно в строки. Они начинаются с буквы f перед открывающей кавычкой и могут содержать выражения в фигурных скобках {} для вставки значений. Были введены с версией Python 3.6. В более старых версиях 2.1, 2.2, и других данная конструкция работать не будет;

```
# F-строки исп-ся исключительно в функции print()
a = 10
b = 5

# простой вывод переменной
print(f"{a}") # 10

# вывод переменной с текстом
print(f"значение a: {a}") # значение a: 10

# операция суммы пример
print(f"сумма: {a+b}") # сумма: 15

# без лишнего текста
print(f"{a+b}") # 15
```

```
# БЕЗ F-СТРОК
a = 10
b = 5

# простой вывод переменной
print(a) # 10

# вывод переменной с текстом
print("значение a: ", a) # значение a: 10

# операция суммы переменных пример
print("сумма: ", (a + b)) # сумма: 15

# без лишнего текста
print(a + b) # 15
```

6. (расписанный 6 вопрос)

Переменные и типы данных. Комментарии в Python, Docstring, Примеры.

(часть из 3го вопроса)

Комментарии в Python используются для объяснения кода и делают его более понятным для других разработчиков (и для вас в будущем). Комментарии игнорируются интерпретатором Python при выполнении программы.

Комментарии можно разделить на:

- **Однострочные комментарии** начинаются с символа # и продолжаются до конца строки.
- **Многострочные комментарии** обычно оформляются с использованием тройных кавычек ("""" или "") и используются как документация или временное исключение блоков кода.

Docstring — это специальный тип комментария в Python, который предназначен для документации функций, классов и модулей. Он помогает описать, что делает функция или класс, какие аргументы принимает и какие значения возвращает. Фактически это и есть многострочный комментарий просто с более широкими возможностями.

Плюсы комментариев

1. Пояснение кода;
2. Улучшение читаемости;
3. Помощь в отладке;
4. Документация (использование docstring);
5. Контроль версий;

Минусы комментариев

1. Источник ошибок;
2. Дублирование информации;
3. Несоответствие коду;
4. Отвлечение от самого кода;
5. Неэффективность документации;

(Примеры можно привести для переменных и также примеры комментариев...)

7. (расписанный 7 вопрос)

Переменные. Типы данных. Функция type(). Преобразование типов (int(), ..). Преобразование типов (list(), dict(), tuple(), set()). Примеры.

(часть из 3го вопроса)

Функция `type()` в Python является встроенной функцией, которая позволяет получить тип данных объекта или переменной. Она чрезвычайно полезна для диагностики и отладки кода, а также для проверки типов данных перед выполнением определенных операций или функций.

Пример:

```
variable = 42
print(type(variable)) # <class 'int'>
```

```
float_variable = 3.14
print(type(float_variable)) # <class 'float'>
```

Это может быть полезно, например, при отладке или при написании кода, который должен различать разные типы данных.

Преобразование типов данных в программировании относится к процессу изменения типа значения одного объекта на другой тип данных.

Неявное преобразование происходит автоматически интерпретатором Python в контексте выражений. Например, операции между целыми числами и числами с плавающей запятой могут привести к неявному преобразованию одного из операндов.

Явное преобразование выполняется с помощью явного вызова функций преобразования типов, что позволяет явно указать, какой тип данных должен иметь объект.

Функции для преобразований:

Функция `int()` используется для преобразования значения в целочисленный тип данных. *Например, `int(3.14)` вернет значение 3, а `int("5")` вернет значение 5.*

Функция `float()` используется для преобразования значения в числовый тип с плавающей точкой. *Например, `float(5)` вернет значение 5.0, а `float("2.7")` вернет значение 2.7.*

Функция `str()` используется для преобразования значения в строковый тип данных. *Например, `str(42)` вернет строку "42", а `str(3.14)` вернет строку "3.14".*

Функция `bool()` используется для преобразования значения в логический тип данных. Любое ненулевое числовое значение или непустая строка будет преобразована в `True`, а ноль или пустая строка будут преобразованы в `False`.

Функции `list`, `dict`, `tuple`, `set`:

- `list()` - используется для создания списка из итерируемого объекта, такого как строка, кортеж или другой список. Но также может служить как функция преобразования какого-то объекта в список.
- `dict()` - может служить для преобразования в словарь;
- `tuple()` - может служить для преобразования в кортеж;
- `set()` - может служить для преобразования в множество;

Примеры преобразований:

```
# Преобразование строки в список символов
s = "hello"
char_list = list(s)
print(char_list) # ['h', 'e', 'l', 'l', 'o']
```

```
# Преобразование кортежа в список
t = (1, 2, 3)
list_from_tuple = list(t)
print(list_from_tuple) # [1, 2, 3]
```

```
# Создание списка на основе другого списка
original_list = [4, 5, 6]
copied_list = list(original_list)
print(copied_list) # [4, 5, 6]
```

```
# Создание словаря из списка кортежей (пар ключ-значение)
tuple_list = [('a', 1), ('b', 2), ('c', 3)]
dict_from_tuples = dict(tuple_list)
print(dict_from_tuples) # {'a': 1, 'b': 2, 'c': 3}
```

```
# Преобразование списка в кортеж
numbers_list = [1, 2, 3]
tuple_from_list = tuple(numbers_list)
print(tuple_from_list) # (1, 2, 3)
```

```
# Создание множества из списка
numbers_list = [1, 2, 3, 2, 1]
unique_numbers_set = set(numbers_list)
print(unique_numbers_set) # {1, 2, 3}
```

8. (расписанный 8 вопрос)

Условная конструкция *IF-ELSE*. Зачем нужен *ELIF*? С примерами.

Условие - это как "если-то". В программировании или математике это способ проверить, выполняется ли какое-то условие, и в зависимости от этого делать разные вещи. Например, "если температура больше 25 градусов, то надень футболку, иначе возьми куртку".

То есть - **Условный оператор в программировании** представляет собой конструкцию, которая позволяет выполнять различные блоки кода в зависимости от выполнения определенного условия. **Условие** - выражение, которое вычисляется как True или False. Оно указывает, какое действие нужно выполнить.

Представь, что ты принимаешь решения в повседневной жизни. Например, ты решаешь, идти ли гулять в парк, если погода хорошая. Если погода хорошая, ты идешь гулять, в противном случае (иначе) остаешься дома. В программировании аналогичные решения делаются с исп-м условного оператора.

Если заменяется на слово IF

Иначе заменяется на слово ELSE

Ещё есть конструкция Иначе если оно заменяется на слово ELIF

Использование конструкции if-elif-else в Python является ключевым аспектом для выполнения условных операций в зависимости от различных сценариев.

- **if, elif** всегда имеют условие;
- **else** никогда не имеет условия;

Стоит соблюдать основные правила написания:

1. **Один блок if, необязательные elif и else:**
 - if является обязательным для начала конструкции условия.
 - elif (else if) и else являются необязательными и могут быть использованы в зависимости от логики условия.
2. **Последовательность проверки:**
 - Условия в блоке if-elif-else проверяются последовательно сверху вниз.
 - Как только одно из условий истинно, соответствующая ветвь выполняется, и проверка завершается.
3. **Взаимоисключающие условия:**
 - Если условия в if и elif взаимоисключающие (например, проверка на равенство двух разных значений), нет необходимости использовать elif или else.
 - В этом случае достаточно использовать только if.
4. **Использование else:**
 - else используется для выполнения кода, если ни одно из условий if и elif не истинно.
 - else может использоваться один раз в конце конструкции if-elif-else.

Простой пример:

```
x = 5
if x > 10:
    print("x is greater than 10")
else:
    print("x is 10 or less")
```

Еще пример:

```
day = "Sunday"

if day == "Sunday":
    print("It's a holiday!")
elif day == "Saturday":
    print("It's the weekend!")
else:
    print("It's a weekday")
```

9. (расписанный 9-12 вопросы)

Циклы. Виды циклов в Python. Схема работы цикла while/for. Функция range(). Понятие бесконечного цикла. Три важных оператора (break, continue, return). Чем цикличность отличается от не-цикличности? Примеры.

Цикл в программировании - это инструкция, которая говорит компьютеру повторять выполнение определенного блока кода несколько раз. Мы используем циклы, чтобы сделать программы более эффективными и чтобы избежать повторения одних и тех же команд.

В Python есть два основных цикла:

- **Цикл while** выполняет блок кода (набор команд), пока условие истинно. Как только условие становится ложным, цикл завершается.
- **Цикл for** используется для перебора элементов в последовательности (например, в списке). Программа будет выполнять определенные действия для каждого элемента в последовательности.

Синтаксис цикла for

```
for переменная in последовательность:  
    # Блок кода, который будет выполнен несколько раз
```

- **переменная** - это переменная, которая будет использоваться для доступа к элементу
- **последовательность** - это набор элементов, по которым будет выполняться цикл.

Схема работы цикла while:

Начало цикла

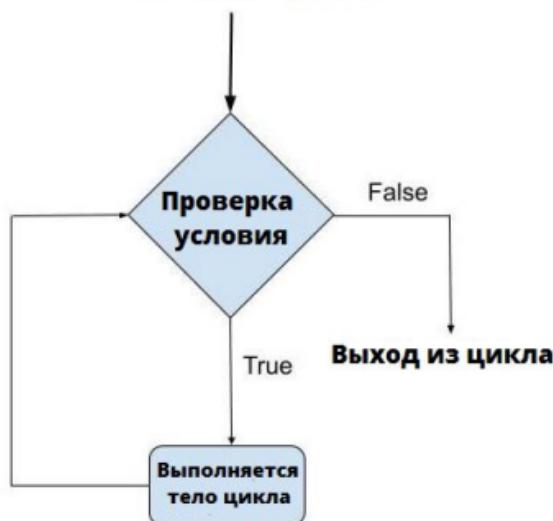


Схема работы цикла for:



`range()` - она генерирует список чисел, который обычно используется для работы с циклом for.

`range(2)` - значит список 0,1

`range(5)` - значит список 0,1,2,3,4

`range(2, 5)` - значит уже от 2 начинаем т.е 2,3,4

`range(2, 5, 2)` - третий аргумент показывает насколько будет увеличена переменная index (стандартно это 1 - и единица не пишется, иными словами это $index += 1$)

Пример кода:

```
for index in range(3):
```

```
    print("Привет, мир!")
```

В этом примере:

1. фраза Hello, world! выводится в консоль ровно три раза.
2. range(3) - значит пройтись от позиции 0 до позиции 3 (не включая цифру 3) т.е три раза 0,1,2 кстати это значения index (счетчика) который переходит внутри цикла с каждой итерацией счетчик увеличивается на 1

Операторы break, continue, return:

`break` полностью выходит из цикла.
`continue` переходит к началу цикла (от слова продолжить)
`return` - оператор возврата чего либо (значений, переменных и тп) из функции или его также можно использовать как жесткий выключатель цикла)

Понятие бесконечного цикла.

Бесконечный цикл в программировании представляет собой циклическую конструкцию, которая выполняется бесконечно, то есть без явного завершения или до тех пор, пока не будет принудительно прервана. Такой тип цикла возникает, когда условие

продолжения выполнения цикла всегда истинно (например, True), или когда условие завершения не определено или не достижимо.

Например:

```
while True:  
    ...  
    while 1:  
        ...
```

Чем цикличность отличается от не-цикличности?

- **Цикличность** означает наличие повторяющихся элементов или действий в структуре или процессе. (циклические данные, циклические алгоритмы и тп)
Использует циклы while или for.
- **Нецикличность** означает отсутствие повторяющихся элементов или действий в структуре или процессе. (что-то линейное уже, единичные процессы)
Использует готовые функции, работает без циклов.

10. (расписанный 13 вопрос)

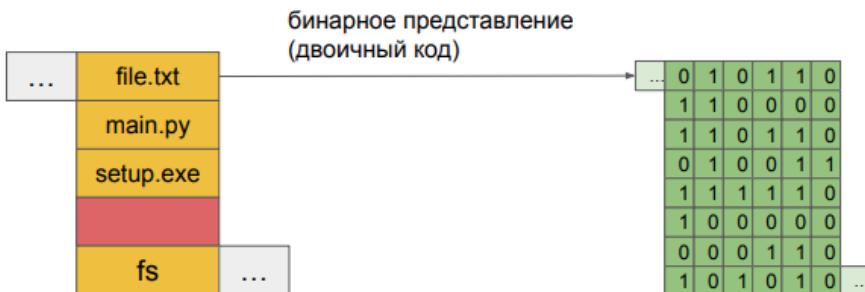
Файловая система. Работа с файлами. Режимы работы с файлами.

Файловая система — это способ организации данных на компьютере или другом устройстве хранения, который позволяет пользователям организовывать, хранить и управлять файлами и каталогами.

Основные компоненты:

- файлы;
- каталоги/директории;
- путь;
- файловые атрибуты;
- и др;

Файлы - это определенное количество информации (программа или данные), имеющее имя и хранящееся в долговременной (внешней) памяти. Файлы на компьютере представлены в виде бинарных данных (двоичный код), которые хранятся на жестком диске или других носителях информации. Каждый файл имеет свое уникальное имя и расширение, которое указывает на тип данных, хранящихся в этом файле.



Работа с файлами - это встроенные функции для чтения и записи файлов.

Вы можете открыть файл с помощью функции `open()`, прочитать его содержимое с помощью метода `read()`, записать данные в файл с помощью метода `write()`, и закрыть файл с помощью метода `close()`.

(запись) - <code>write()</code>	<code>w, a (дозапись)</code>
(прочитать) - <code>read()</code>	<code>r</code>

Режимы работы с файлами.

Режим	Обозначение
'r'	открытие на чтение (является значением по умолчанию).
'w'	открытие на запись, содержимое файла удаляется, если файла не существует, создается новый.
'x'	открытие на запись, если файла не существует, иначе исключение.
'a'	открытие на дозапись, информация добавляется в конец файла.
'b'	открытие в двоичном режиме.
't'	открытие в текстовом режиме (является значением по умолчанию).
'+'	открытие на чтение и запись

Работа с файлами. Пример.

```
python main.py > ...
1  f = open("myfile.txt", "w") # открываем файл для записи
2  f.write("Hello, World!") # записываем строку в файл
3  f.close() # закрываем файл
4
5  f = open("myfile.txt", "r") # открываем файл для чтения
6  print(f.read()) # читаем и печатаем содержимое файла
7  f.close() # закрываем файл
8
9  # новый синтаксис
10 with open("myfile.txt", "r") as f:
11     print(f.read())
```

11. (расписанный 14 вопрос)

*Программирование. Что такое компиляция программы? Понятие алгоритма. PEP. Типы PEP.
(из 1го вопроса)*

PEP в Python означает Python Enhancement Proposal. PEP - это документ, который описывает новые функции Python, процессы и окружения, которые влияют на большую часть сообщества Python.

PEP.

Именование: Используйте CamelCase для имен классов, lowercase_with_underscores для функций и методов, и UPPER_CASE_WITH_UNDERSCORES для констант.

Отступы: Используйте 4 пробела на уровень отступа, а не табуляцию.

Длина строки: Строки должны быть не более 79 символов.

Пробелы: Используйте пробелы вокруг операторов и после запятой для улучшения читаемости.

Импорты: Импорты должны быть на отдельных строках и размещаться в начале файла.

Типы PEP:

- **PEP стандарта:** PEP, которые определяют новые функции языка Python, изменения в существующем поведении или добавление новых модулей в стандартную библиотеку. Примеры: PEP 8 (стиль кодирования), PEP 20 (Zen of Python).
- **PEP процесса:** PEP, описывающие изменения в процессе разработки Python, включая изменения в процедурах принятия решений, методах обратной связи и т.д. Пример: PEP 1 (PEP Purpose and Guidelines).
- PEP информационные;
- PEP экспериментальные;

2 БЛОК.

1. Коллекции в Python. Списки. Срезы. Примеры.

ОБЩЕЕ О КОЛЛЕКЦИЯХ:

Коллекция в Python - это контейнер для хранения и работы с группой элементов одного или разных типов данных, таких как числа, строки, списки, словари и т.д. Коллекции можно изменять, добавлять, удалять элементы, сортировать и выполнять на них различные операции.

Основные типы коллекций в Python:

1. Списки (Lists):

- Упорядоченные коллекции элементов.
- Могут содержать элементы различных типов данных.
- Поддерживают изменяемость (могут быть изменены после создания).
- Используются для хранения и доступа к данным в порядке их добавления.

2. Кортежи (Tuples):

- Упорядоченные коллекции элементов, подобные спискам.
- Неизменяемы (не могут быть изменены после создания).
- Используются для представления неизменяемых последовательностей данных.

3. Множества (Sets):

- Неупорядоченные коллекции уникальных элементов.
- Поддерживают операции над множествами, такие как объединение, пересечение и разность.
- Используются для удаления дубликатов и проверки наличия элементов.

4. Словари (Dictionaries):

- Коллекции, хранящие пары ключ-значение.
- Позволяют быстрый доступ к значениям по ключу.
- Ключи должны быть уникальными и неизменяемыми, а значения могут быть изменяемыми или неизменяемыми.

Цели использования коллекций в Python:

- Хранение данных;
- Манипуляции с данными;
- Итерация и доступ к элементам;
- Управление памятью и производительностью;

СПИСКИ;

Основные методы работы со списком:

- **append()**: Добавляет элемент в конец списка.
- **extend()**: Расширяет список, добавляя элементы другого списка.
- **insert()**: Вставляет элемент в указанную позицию списка.
- **remove()**: Удаляет первое вхождение указанного элемента из списка.
- **pop()**: Удаляет элемент с указанным индексом и возвращает его.
- **index()**: Возвращает индекс первого вхождения указанного элемента.
- **count()**: Возвращает количество вхождений указанного элемента в список.
- **sort()**: Сортирует элементы списка по возрастанию (или по ключу).
- **reverse()**: Изменяет порядок элементов списка на обратный.
- **clear()**: Удаляет все элементы из списка.

также для всех коллекций применима функция `len()`;

В Python оператор [] используется для доступа к элементам в списках, кортежах, строках и других последовательностях. В случае со списками (list), основные функции оператора [] включают:

1. **Доступ к элементу по индексу**: Можно получить доступ к элементу списка, указав его индекс в квадратных скобках. Индексы начинаются с 0 для первого элемента списка.

```
a = ['a', 'b', 'c', 'd']
print(a[0]) # Выводит 'a'
print(a[2]) # Выводит 'c'
```

2. **Изменение элемента по индексу**: Элементы списка можно изменять, присвоив им новое значение через оператор []. Например:

```
a[1] = 'x'
print(a) # Выводит ['a', 'x', 'c', 'd']
```

3. **Извлечение среза (slice)**: Оператор [] также поддерживает извлечение срезов (slice), позволяя получать подмножества элементов списка.

Оператор in используется для проверки принадлежности элемента к последовательности (например, списку, кортежу, строке и т. д.). Он возвращает True, если элемент присутствует в последовательности, и False, если нет.

```
my_list = ['apple', 'banana', 'cherry']
print('banana' in my_list) # Выводит True
print('orange' in my_list) # Выводит False
```

Оператор del в Python удаляет объекты (например, переменные или элементы списка) из памяти компьютера, тем самым освобождая память для других операций.

Пример работы со списками. (заполнение списка)

```
# Создаем пустой список
numbers = []

# Просим пользователя ввести три числа
for i in range(3):
    num = int(input("Введите число: "))
    # Добавляем число в список
    numbers.append(num)

# Выводим список на экран
print("Список чисел:", numbers)
```

Пример работы со списками. (удалить i-й элемент из списка)

```
# Создаем список
my_list = [1, 2, 3, 4, 5]

# Удаляем элемент с индексом 2
del my_list[2]

# Печатаем список после удаления
print(my_list)

# Результат: [1, 2, 4, 5]
```

(ну и другие примеры)

СРЕЗЫ:

Срез (slice) в Python — это операция извлечения подстроки из последовательности (например, строки, списка, кортежа), указывая начальный и конечный индексы.

Синтаксис среза. sequence[start:stop:step]

где `sequence` — последовательность, к которой применяется операция среза,
`start` — индекс элемента, с которого начинается срез,
`stop` — индекс элемента, до которого идет срез (*не включая элемент с этим индексом*),
`step` (*необязательный*) — шаг, с которым нужно пройти по последовательности, чтобы выбрать элементы для среза.

Простой пример среза.

*syntax: sequence[start:stop:step]

```
s = "Hello, world!"  
print(s[1:5]) # "ello"  
print(s[:5]) # "Hello,"  
print(s[7:]) # "world!"  
print(s[::-2]) # "Hlo ol!"
```

(в срезах - также можно примеры на списки и тп добавить)

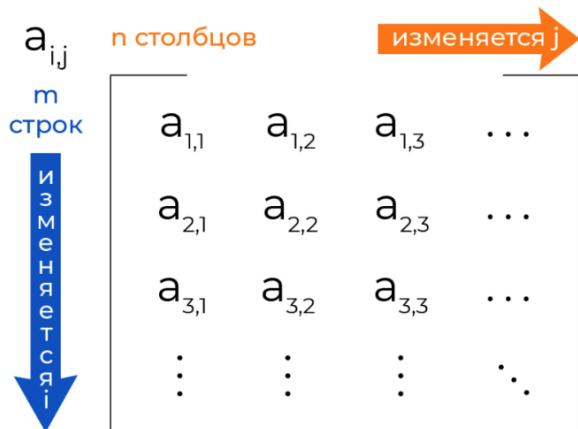
2. Коллекции в Python. Списки. Матрица элементов. Типы матриц. Примеры.

(из 1го вопроса)

Матрица: Прямоугольная таблица чисел, расположенных в виде строк и столбцов.

Элемент матрицы: Каждый индивидуальный номер в матрице, обозначаемый как a_{ij} , где i - номер строки, j - номер столбца. **Размер матрицы:** Определяется количеством строк и столбцов, например, матрица размера $M \times N$ имеет M строк и N столбцов.

Матрица m на n



Типы матриц:

- **Квадратная матрица:** Матрица, у которой количество строк равно количеству столбцов ($n \times n$).

$$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 5 & 7 \end{pmatrix}$$

- **Единичная матрица:** Квадратная матрица с единицами на главной диагонали и нулями в остальных местах.

$$E_1 = (1), E_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, E_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- **Нулевая матрица:** Матрица, все элементы которой равны нулю.

$$V = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

- **Транспонированная матрица:** Матрица, полученная заменой строк на столбцы исходной матрицы.

$$a) A^T = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}^T = \begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix}^T$$

- **Диагональная матрица:** Все элементы вне главной диагонали равны нулю.

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 11 \end{pmatrix}$$

- **Треугольная матрица:** Верхняя или нижняя матрица с нулями ниже или выше главной диагонали соответственно.

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 0 & 4 & 5 \\ 0 & 0 & 6 \end{pmatrix} \quad B = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 3 & 0 \\ 4 & 5 & 6 \end{pmatrix}$$

1) Создание матрицы 2x2

```
A = [
    [1, 2],
    [3, 4]
]
```

2) Создание матрицы 3x3

```
A = [
    [1, 2, 1],
    [-1, 0, 1],
    [2, 1, 4]
]
```

4) Вывод матрицы на экран (построчно).

```
# Вывод матрицы
for row in A:
    print(row)
```

3. Коллекции в Python. Списки. Матрица элементов. Матричные алгоритмы. Примеры.

(из 1го и 2го вопроса)

МАТРИЧНЫЕ АЛГОРИТМЫ:

1) Перемножение матриц.

- Убедитесь, что количество столбцов первой матрицы равно количеству строк второй матрицы.
- Далее произведение матриц $C = A * B$; $C_{ij} = \sum_{k=1}^n A_{ik} * B_{kj}$
 - A_{ik} — элемент, находящийся на пересечении i -ой строки и k -ого столбца матрицы А.
 - B_{kj} — элемент, находящийся на пересечении k -ой строки и j -ого столбца матрицы В.

Пример перемножения двух матриц размера 2x2:

$$\begin{aligned} BC &= \begin{pmatrix} -28 & 93 \\ 38 & -126 \end{pmatrix} \begin{pmatrix} 7 & 3 \\ 2 & 1 \end{pmatrix} = \begin{pmatrix} -28 \times 7 + 93 \times 2 & -28 \times 3 + 93 \times 1 \\ 38 \times 7 - 126 \times 2 & 38 \times 3 - 126 \times 1 \end{pmatrix} \\ &= \begin{pmatrix} -10 & 9 \\ 14 & -12 \end{pmatrix} \end{aligned}$$

2) Сложение матриц.

- Матрицы должны иметь одинаковый размер.
- Далее сложение матриц $C = A + B$; $C_{ij} = A_{ij} + B_{ij}$.

Простой пример сложения двух матриц:

$$\begin{aligned} A + B &= \begin{bmatrix} 2 & -3 & 1 \\ 5 & 4 & -2 \end{bmatrix} + \begin{bmatrix} 4 & 2 & -5 \\ -4 & 1 & 3 \end{bmatrix} = \\ &= \begin{bmatrix} 2+4 & -3+2 & 1-5 \\ 5-4 & 4+1 & -2+3 \end{bmatrix} = \begin{bmatrix} 6 & -1 & -4 \\ 1 & 5 & 1 \end{bmatrix} \end{aligned}$$

3) Вычисление определителя матрицы.

Определитель матрицы — это значение, которое характеризует квадратную матрицу и дает информацию о ее свойствах, таких как обратимость и линейную независимость строк или столбцов. Обозначается как \det . Например, $\det A = \dots$, $\det B = \dots$; Но также может обозначаться как Δ_2 , Δ_1 , Δ_3 — цифры 2; 1; 3 это размер матрицы

1. Для 2×2 матрицы $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$, $\det(A) = ad - bc$.
2. Для $n \times n$ матрицы, определитель рассчитывается рекурсивно с использованием разложения по строке или столбцу.

Простой пример:

$$\Delta_2 = |A| = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11} \cdot a_{22} - a_{12} \cdot a_{21}.$$

$$\Delta_2 = |A| = \begin{vmatrix} -2 & 3 \\ -4 & 5 \end{vmatrix} = -2 \cdot 5 - 3 \cdot (-4) = -10 + 12 = 2.$$

4) Сумма элементов строк матрицы.

1. Для матрицы A размера $m \times n$:

$$\text{sum}_i = \sum_{j=1}^n A_{ij}$$

где sum_i — сумма элементов i -ой строки.

Простой пример суммы:

$$A = \begin{pmatrix} 1 & 2 & 3 \\ -1 & 0 & 1 \\ 2 & -1 & 4 \end{pmatrix}$$
$$\text{sum } A = \begin{pmatrix} 1+2+3 \\ -1+0+1 \\ 2-1+4 \end{pmatrix} = \begin{pmatrix} 6 \\ 0 \\ 5 \end{pmatrix}$$

4. Коллекции в Python. Списки, кортежи. Чем матрица от обычного списка отличается? Сфера применения матриц. Определитель матрицы. Примеры.

(из 1го и 2го вопроса)

Чем матрица от обычного списка отличается?

- **Обычный список:** Это структура данных, которая представляет собой упорядоченную коллекцию элементов. Элементы могут быть любых типов данных (числа, строки, другие списки и т.д.) и хранятся в линейной последовательности. Элементы в списке индексируются одним целым числом (индексом), начиная с нуля. Доступ к элементу происходит по его индексу. Это одномерная структура данных, в которой элементы хранятся последовательно в одной линии. Используется для хранения и обработки коллекций данных.
- **Матрица:** Это двумерная структура данных, организованная в виде таблицы, состоящей из строк и столбцов. Каждый элемент матрицы располагается в определенной позиции, задаваемой двумя индексами: индекс строки и индекс столбца. Элементы матрицы индексируются двумя целыми числами: индекс строки и индекс столбца. Например, элемент в строке i и столбце j матрицы A обозначается как $A[i][j]$. Это двумерная структура данных. Используется для представления двумерных данных, таких как таблицы данных, изображения, графики и т.д., где важны их расположение в строках и столбцах.

Функции и сферы применения матриц:

- **Линейная алгебра:** Решение систем линейных уравнений, нахождение собственных значений и собственных векторов.

$$\det \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} = a_{11}a_{22} - a_{21}a_{12}$$
$$\det \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{32}a_{21} - a_{11}a_{23}a_{32} - a_{22}a_{13}a_{31} - a_{33}a_{12}a_{21}$$
$$\left(\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \right)^{-1} = \frac{1}{a_{11}a_{22} - a_{12}a_{21}} \begin{pmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{pmatrix}$$

- **Компьютерная графика:** Преобразования изображений, масштабирование, поворот и трансляция.



- **Машинное обучение:** Представление и обработка данных, операции с большими наборами данных.



- **Физика и инженерия:** Моделирование физических процессов и систем.
- А также программирование...

Определитель матрицы — это значение, которое характеризует квадратную матрицу и дает информацию о ее свойствах, таких как обратимость и линейную независимость строк или столбцов. Обозначается как \det . Например, $\det A = \dots$, $\det B = \dots$; Но также может обозначаться как Δ_2 , Δ_1 , Δ_3 — цифры 2; 1; 3 это размер матрицы

5. Коллекции в Python. Списки, кортежи. Отличия. Примеры. (из 1го вопроса)

Кортежи **не имеют методов** для изменения содержимого из-за их неизменяемости, но вы можете использовать общие функции, такие как `len(tuple)` для получения длины кортежа и `tuple.count(x)` для подсчета количества раз, которое x встречается в кортеже.

Отличия списков и кортежей:

Свойство/Признак	Списки	Кортеж
Изменяемость	Изменяемы: Элементы списка можно изменять, добавлять и удалять после его создания.	Неизменямы: Элементы кортежа не могут быть изменены после создания. Однако можно создать новый кортеж с измененными или добавленными элементами.
Скорость работы	более медленны с большими данными;	быстрее списком (если большие данные)
Использование	Используются там, где требуется изменяемая последовательность элементов. Часто используются для хранения данных одного типа или разнородных типов. Примеры: хранение данных в структурах данных, стеки, очереди и т. д.	Используются там, где данные должны оставаться неизменяемыми. Часто используются для группировки данных различных типов, когда порядок элементов важен. Примеры: использование в качестве ключей словаря, возвращаемых значений функций, фиксированных структур данных и т. д.
Синтаксис	Используется квадратные скобки [] для создания: <code>lst = [1, 2, 3]</code>	Используется круглые скобки () для создания: <code>tpl = (1, 2, 3)</code>

6. Коллекции в Python. Списки, срезы. Типы циклов пробежки по списку. Примеры.

(из 1го вопроса)

(рассказать кратко о циклах while и for)

Типы циклов пробежки по списку.

1. Использование цикла while:

```
A = [1, 2, 3, 4, 5]
index = 0
while index < len(A):
    print(A[index])
    index += 1
```

2. Использование цикла for:

- с range() - пробежка по индексам списка.

```
A = ['a', 'b', 'c', 'd', 'e']

for i in range(len(A)):
    print(f"Индекс {i}: {A[i]}")
```

- без range() - пробежка по значениям списка.

```
A = ['a', 'b', 'c', 'd', 'e']

for item in A:
    print(item)
```

7. Коллекции в Python. Списки, множества. Отличия. Примеры. (из 1го вопроса)

ОТЛИЧИЯ

Свойство/Признак	Списки	Множество
Изменяемость	Изменяемы: Элементы списка можно изменять, добавлять и удалять после его создания.	Изменяемые: можно добавлять и удалять элементы.
Скорость работы	Более медленные при операциях поиска и проверки наличия элемента.	Быстрые при операциях поиска и проверки наличия элемента из-за использования хэш-таблиц.
Использование	Используются для хранения упорядоченных коллекций элементов. Поддерживают дублирование элементов.	Используются для хранения уникальных элементов без определенного порядка. Не поддерживают дублирование элементов.
Синтаксис	Используется квадратные скобки [] для создания: lst = [1, 2, 3]	Объявляются с помощью фигурных скобок: {1, 2, 3} или функции set(): set([1, 2, 3])

(ну и примеры списков и множеств)

8. Коллекции в Python. Списки, словари. Отличия. Примеры. (из 1го вопроса)

ОТЛИЧИЯ:

Свойство/Признак	Списки	Словари
Изменяемость	Изменяемы: Элементы списка можно изменять, добавлять и удалять после его создания.	Изменяемые: можно добавлять, удалять, изменять пары "ключ-значение".
Скорость работы	Быстрый доступ по индексу, медленный поиск элементов.	Быстрый доступ по ключу, быстрый поиск по ключу.
Использование	Используются для хранения упорядоченных коллекций элементов. Поддерживают дублирование элементов.	Используются для хранения пар "ключ-значение", где ключи уникальны.
Синтаксис	Используется квадратные скобки [] для создания: lst = [1, 2, 3]	Объявляются с помощью фигурных скобок: {'key1': 'value1', 'key2': 'value2'}

```
d = {'name': 'Alice', 'age': 25}
d['city'] = 'New York' # {'name': 'Alice', 'age': 25, 'city': 'New York'}
d['age'] = 26          # {'name': 'Alice', 'age': 26, 'city': 'New York'}
```

(ну и примеры списков и словарей)

Чем матрица от обычного списка отличается?

- **Обычный список:** Это структура данных, которая представляет собой упорядоченную коллекцию элементов. Элементы могут быть любых типов данных (числа, строки, другие списки и т.д.) и хранятся в линейной последовательности. Элементы в списке индексируются одним целым числом (индексом), начиная с нуля. Доступ к элементу происходит по его индексу. Это одномерная структура данных, в которой элементы хранятся последовательно в одной линии. Используется для хранения и обработки коллекций данных.
- **Матрица:** Это двумерная структура данных, организованная в виде таблицы, состоящей из строк и столбцов. Каждый элемент матрицы располагается в определенной позиции, задаваемой двумя индексами: индекс строки и индекс столбца. Элементы матрицы индексируются двумя целыми числами: индекс строки и индекс столбца. Например, элемент в строке i и столбце j матрицы A обозначается как $A[i][j]$. Это двумерная структура данных. Используется для представления двумерных данных, таких как таблицы данных, изображения, графики и т.д., где важны их расположение в строках и столбцах.

ПРИМЕРЫ ЗАДАЧ

Примеры задач для самоподготовки:

1. Дан список (который задает пользователь - вводит с использованием спец. функции заполнения), найти минимальный положительный элемент в этом списке.

2. Дан список (который задает пользователь - вводит с использованием спец. функции заполнения), найти минимальный положительный четный элемент в этом списке без использования спец. функций (min, max. и др).

3. Даны две переменные a и b найти среднее арифметическое значение (avg) между a и b . Затем создать кортеж состоящий из 5 элементов, каждый из которых это самое среднее значение (avg). Вывести полученный кортеж на экран.

4. Даны две переменные a и b . Вычислить значение выражения и вывести его на экран.

$$\frac{a * b + 100 * (b - a * 2)}{a \% b + 1 * 3 * 4 * b^a}$$

5. Даны две переменные a и b . Вычислить значение выражения и вывести его на экран. (И - логическое "И")

$$\frac{2*a^{b+1} + 99 * (b \text{ И } a)}{b^{a-1} * b^{a-2} * 345 + 100}$$

6. Дан список из чисел и строк. Найти сумму четных чисел в списке и вывести результат на экран.

Ist = [1, "AA", 45.5, "B", 45, 55, 2, "FFF", "L", 3.14, 111, 0, -1, -2, 12]

7. Найти сумму трех переменных (типа float) поделенная на 1234. Вывести результат на экран.

могут быть и другие любые задачи на пройденные темы (матриц не будет)!

решение 1й задачи:

```

lst = []
length = int(input("enter length: "))
for i in range(length):
    item = int(input(f"enter element ({i}): "))
    lst.append(item)

min_positive = None
for item in lst:
    if item > 0:
        if min_positive is None or item < min_positive:
            min_positive = item

if min_positive is not None:
    print("Минимальный положительный элемент:", min_positive)
else:
    print("В списке нет положительных элементов.")

```

решение 4й задачи:

```

a = float(input("Введите значение a: "))
b = float(input("Введите значение b: "))

numerator = a * b + 100 * (b - a**2)
denominator = a % b + 1 * 3 * 4 * b**a

if denominator == 0:
    print("Знаменатель равен нулю, вычисление невозможно.")
else:
    result = numerator / denominator
    print("Результат выражения:", result)

```

решение 6й задачи:

```

lst = [1, "AA", 45.5, "B", 45, 55, 2, "FFF", "L", 3.14, 111, 0, -1, -2, 12]
sum_even = 0
for item in lst:
    if type(item) == int or type(item) == float:
        if item == int(item) and int(item) % 2 == 0:
            sum_even += item
print(sum_even)

```