Оглавление

[Вопросы, часто встречающиеся на собеседованиях 2](#_Toc144739479)

[Python: 2](#_Toc144739480)

[Git 15](#_Toc144739481)

[Docker 16](#_Toc144739482)

[Базы данных 17](#_Toc144739483)

[SQL 18](#_Toc144739484)

[Backend 19](#_Toc144739485)

[Тестирование 20](#_Toc144739486)

[Кодинг: 21](#_Toc144739487)

[Задача на временные метки 22](#_Toc144739488)

**Вопросы, часто встречающиеся на собеседованиях**

**Python:**

1. **Что такое интерпретируемый язык? Что такое интерпретатор и как он работает в Python?**

Python – это интерпретируемый язык программирования. Это означает, что код исполняется «на лету», строка за строкой. К примеру, используя Python Shell, можно вводить код по одной строке, и каждая из них будет тут же выполнена интерпретатором.

Интерпретатор Питона работает следующим образом:

* сперва исходный код, который написан программистом и хранится в файлах с расширением .py, преобразуется в байт-код (файлы с расширением .pyc). Байт-код – это промежуточный слой между человеко-читаемым кодом и кодом на языке процессора.
* затем байт-код исполняется виртуальной машиной.

1. **Что такое статическая и динамическая типизация и в чем это проявляется в Python?**

* Статическая типизация — у каждой именованной сущности (переменной, функции…) есть свой фиксированный тип, он не меняется в процессе выполнения программы. Примеры: Си, C++, Java, Haskell, Rust, Go.
* Динамическая — тип переменной/функции известен только во время выполнения программы. Примеры: Scheme, JavaScript, Python.

Python поддерживает динамическую типизацию, то есть тип переменной определяется только во время исполнения. Поэтому вместо «присваивания значения переменной» лучше говорить о «связывании значения с некоторым именем».

1. **Какие типы данных в Python вам известны?**

* изменяемые (множества-внутри множества только неизм. данные, списки, словари-только значения, булево);
* неизменяемые (кортежи, строки, числа);
* упорядоченные (списки, строки, кортежи, словари);
* неупорядоченные (множества)

1. **Чем кортеж отличается от списка?**

Кортеж неизменяемая упорядоченная коллекция данных. ()

Список изменяемый. []

1. **В каких случаях лучше использовать кортеж, а в каких - список?**

* кортежи используют если нам нужно чтобы не было доступа к данным, и мы не могли их изменять. Списки, наоборот, поддерживают функцию вставки.
* Также стоит учитывать, что списки обычно занимают больше памяти, чем кортежи, из-за того, что списки могут изменяться в процессе работы программы.

1. **Как вы объясните человеку, что такое множество?**

Множество – это изменяемая неупорядоченная коллекция уникальных данных. Внутри множества находятся только неизменяемый тип данных.

{x, y, …}

1. **Что такое словари и как они работают?**

Словари – это изменяемый коллекция данных. В словарях можно изменить только значения. {key: value}. По ключу словаря можно получить значение.

1. **Как работают типы данных хэшмап, связанный список, двусвязный список, массив?**

* Хешмап тоже самое что и словарь и поддерживает те же функции: быстрая вставка, извлечение, удаление данных.
* [Связанный список](https://translated.turbopages.org/proxy_u/en-ru.ru.0cd5dcc1-64f5f70e-f0095ea3-74722d776562/https/www.geeksforgeeks.org/what-is-linked-list/) — это тип линейной структуры данных, похожий на массивы. Это набор узлов, которые связаны друг с другом. Узел содержит две вещи: первая — это данные, а вторая - ссылку, которая соединяет его с другим узлом.
* В односвязном списке каждый узел списка имеет два компонента: фактическое значение узла и ссылку на следующий узел в связанном списке. В двусвязном списке каждый узел имеет три компонента: значение узла, ссылку на предыдущий узел и ссылку на следующий узел. Для начального узла двусвязного списка ссылка на предыдущий узел равна нулю.
* В языке Python как таковых массивов нет. Вместо этого для хранения группы однотипных (но и не только однотипных) объектов используют списки - объекты типа list. Отличие списков от массивов заключается в том, что список — это динамическая структура.

1. **Что такое О (о-большое)? Зачем и как используется?**

Big O нотация нужна для описания сложности алгоритмов.

O (1), O(n) – линейная зависимость, O (log n) – логарифмическая зависимость, O (n\*log n) – логарифмическая зависимость с константой, O(n^2) – квадратичная зависимость, O(n!) - факториальная зависимость.

1. **Какая сложность вставки, извлечения, поиска элементов в словаре и списке?**

O (1) и там и там.

1. **Что такое функции и зачем они нужны?**

Функции в Python представляют собой фрагменты кода в блоке, который имеет назначенное имя. Функции принимают ввод, осуществляют вычисления либо какое-нибудь действие и возвращают вывод. И, разумеется, функции упрощают работу с кодом, делая возможным его повторное использование.

1. **Какие бывают аргументы у функций?**

Функция может принимать произвольное количество аргументов или не принимать их вовсе. Также распространены функции с произвольным числом аргументов, функции с позиционными и именованными аргументами, обязательными и необязательными.

1. **Что такое распаковка коллекций?**

* Распаковка (unpacking, также называемая Деструктуризация) представляет разложение коллекции (кортежа, списка и т.д.) на отдельные значения.
* Так, как и многие языки программирования, Python поддерживает концепцию множественного присваивания. Например:

|  |  |
| --- | --- |
| 1  2  3 | x, y = 1, 2  print(x)    # 1  print(y)    # 2 |

1. **Что такое исключения? Какие они бывают? Как их использовать?**

Исключения вызываются, когда происходит ошибка при выполнении кода.

Ключевые слова: try, except, finally.

1. **Что такое ООП? На каких концепциях стоит ООП?**

Объектно-ориентированное программирование (ООП) — это парадигма программирования, где различные компоненты компьютерной программы моделируются на основе реальных объектов. **Объект** — это что-либо, у чего есть какие-либо характеристики и то, что может выполнить какую-либо функцию.

Абстракция, Наследование, Инкапсуляция, Полиморфизм.

1. **Объяснить понятие класса и объекта (экземпляра).**

* Класс в Python — это логическая группа данных и функций. Он дает возможность создавать структуры данных, которые содержат произвольный контент и, следовательно, легко доступны.
* Объект также называется экземпляром. Тем не менее, процесс создания объекта класса называется инициализация. В Python, чтобы создать объект класса, нам просто нужно вписать название класса, с последующими открывающимися и закрывающимися скобками.

1. **Что такое инкапсуляция? Накодить пример.**

Инкапсуляция – механизм сокрытия деталей реализации класса от других объектов. Достигается путем использования модификаторов доступа public, private и protected, которые соответствуют публичным, приватным и защищенным атрибутам.

Сделаем атрибуты title, author и isbn класса Book приватными – теперь доступ к ним возможен только внутри класса:

class Book:

def \_\_init\_\_(self, title, author, isbn):

self.\_\_title = title # приватный

self.\_\_author = author # приватный

self.\_\_isbn = isbn # приватный

Чтобы получить доступ к этим атрибутам извне класса, мы определяем методы getter и setter, которые обеспечивают контролируемый доступ к атрибутам:

def get\_title(self):

return self.\_\_title

def set\_title(self, title):

self.\_\_title = title

def get\_author(self):

return self.\_\_author

def set\_author(self, author):

self.\_\_author = author

def get\_isbn(self):

return self.\_\_isbn

1. **Что такое наследование? Накодить пример.**

Наследование – процесс создания нового класса на основе существующего класса. Новый класс, называемый подклассом или производным классом, наследует свойства и методы существующего класса, называемого суперклассом или базовым классом.

Для иллюстрации концепции наследования мы определим класс Publication, который имеет свойства, общие для всех публикаций – title, author и year, а также общий метод display():

class Publication:

def \_\_init\_\_(self, title, author, year):

self.title = title

self.author = author

self.year = year

def display(self):

print("Название:", self.title)

print("Автор:", self.author)

print("Год выпуска:", self.year)

Теперь создадим два подкласса Book и Magazine, которые наследуют все свойства и методы от класса Publication, и кроме того, имеют свои атрибуты. Подкласс Book добавляет свойство isbn и переопределяет метод display() для включения свойства isbn. Подкласс Magazine добавляет свойство issue\_number (номер выпуска) и переопределяет методdisplay() для включения свойства issue\_number:

class Book(Publication):

def \_\_init\_\_(self, title, author, year, isbn):

super().\_\_init\_\_(title, author, year)

self.isbn = isbn

def display(self):

super().display()

print("ISBN:", self.isbn)

class Magazine(Publication):

def \_\_init\_\_(self, title, author, year, issue\_number):

super().\_\_init\_\_(title, author, year)

self.issue\_number = issue\_number

def display(self):

super().display()

print("Номер выпуска:", self.issue\_number)

1. **Бывает ли в Python множественное наследование? Как ты относишься к этому факту?**

Да, бывает. При множественном наследовании надо обращать внимание на инициализатор родительских классов. Смотреть что мы передаем в его аргументы. Миксины в целом помогают сделать код более читабельным и структурированным.

1. **Если у нескольких классов-родителей есть методы с одинаковыми названиями, то какой из них будет вызван в наследнике?**

Будет вызван тот метод, родительский класс, которого унаследован первый.

1. **Что такое полиморфизм? Накодить пример.**

Полиморфизм – это возможность работы с совершенно разными объектами (языка Python) единым образом.

**class** **Cat**:

**def** **\_\_init\_\_**(self, name, age):

self.name = name

self.age = age

**def** **info**(self):

print(f"I am a cat. My name is {self.name}. I am {self.age} years old.")

**def** **make\_sound**(self):

print("Meow")

**class** **Dog**:

**def** **\_\_init\_\_**(self, name, age):

self.name = name

self.age = age

**def** **info**(self):

print(f"I am a dog. My name is {self.name}. I am {self.age} years old.")

**def** **make\_sound**(self):

print("Bark")

cat1 = Cat("Kitty", 2.5)

dog1 = Dog("Fluffy", 4)

**for** animal **in** (cat1, dog1):

animal.make\_sound()

animal.info()

animal.make\_sound()

**Вывод:**

Meow

I am a cat. My name **is** Kitty. I am 2.5 years old.

Meow

Bark

I am a dog. My name **is** Fluffy. I am 4 years old.

Bark

Здесь мы создали два класса Cat и Dog. У них похожая структура и они имеют методы с одними и теми же именами info() и make\_sound().

Однако, заметьте, что мы не создавали общего класса-родителя и не соединяли классы вместе каким-либо другим способом. Даже если мы можем упаковать два разных объекта в кортеж и итерировать по нему, мы будем использовать общую переменную animal. Это возможно благодаря полиморфизму.

1. **Что такое staticmethod?**

* @staticmethod – используется для создания метода, который ничего не знает о классе или экземпляре, через который он был вызван. Он просто получает переданные аргументы, без неявного первого аргумента, и его определение неизменяемо через наследование.
* Проще говоря, @staticmethod – это вроде обычной функции, определенной внутри класса, которая не имеет доступа к экземпляру, поэтому ее можно вызывать без создания экземпляра класса.
* Как мы уже говорили, мы можем получить доступ к статическому методу класса без создания экземпляра.

Myclass.staticmethod()

1. **Чем staticmethod отличается от простой функции?**

Staticmethod привязан к классу по смыслу и не загрязняет пространство имен. Это делается для удобного чтения и закрытого доступа к функции при необходимости.

1. **Когда будем использовать staticmethod, а когда простую функцию?**

* Staticmethod привязан к классу по смыслу и не загрязняет пространство имен. Это делается для удобного чтения и закрытого доступа к функции при необходимости.
* Статический метод помогает в достижении инкапсуляции в классе, поскольку он не знает о состоянии текущего экземпляра. Кроме того, статические методы делают код более читабельным и повторно используемым, а также более удобным для импорта по сравнению с обычными функциями, поскольку каждую функцию не нужно отдельно импортировать.

1. **Что такое classmethod? В чём его особенности? Когда применяется?**

* @classmethod – это метод, который получает класс в качестве неявного первого аргумента, точно так же, как обычный метод экземпляра получает экземпляр. Это означает, что вы можете использовать класс и его свойства внутри этого метода, а не конкретного экземпляра.
* Проще говоря, @classmethod – это обычный метод класса, имеющий доступ ко всем атрибутам класса, через который он был вызван. Следовательно, classmethod – это метод, который привязан к классу, а не к экземпляру класса.

Когда использовать classmethod?

@classmethod используется, когда вам нужно получить методы, не относящиеся к какому-либо конкретному экземпляру, но тем не менее, каким-то образом привязанные к классу. Самое интересное в них то, что их можно переопределить дочерними классами.

Поэтому, если вы хотите получить доступ к свойству класса в целом, а не к свойству конкретного экземпляра этого класса, используйте classmethod.

1. **Что такое магические методы и за что они отвечают?**

Магические методы - это методы Python, которые определяют, как ведут себя объекты Python при выполнении над ними обычных операций. Эти методы четко определены с помощью двойного подчеркивания до и после имени метода

1. **Что такое конструктор объектов в Python и из чего он состоит и как работает?**

**Конструктор в Python – это особый тип метода (функции), который используется для инициализации членов экземпляра класса.**

**В Python метод \_\_init \_\_() имитирует конструктор класса. Этот метод вызывается при создании экземпляра класса. Он принимает ключевое слово self в качестве первого аргумента, который позволяет получить доступ к атрибутам или методу класса.**

1. **Почему в методе init есть аргумент self?**

Self – это ссылка на экземпляр класса для дальнейшей передачи его как аргумента при вызове класса.

1. **Что такое приватные и защищенные атрибуты? Как они работают?**

Чтобы регулировать вмешательство во внутреннюю работу объекта, в ООП есть несколько уровней доступа: публичный (public), защищённый (\_protected) и приватный (\_\_private). Защищённые атрибуты и методы можно вызывать только внутри класса и его классов-наследников. Приватные — только внутри класса: даже наследники не имеют доступа к ним.

1. **Можно ли получить доступ к приватным атрибутам в Python? да**

При этом в Python само по себе объявление атрибутов и методов защищёнными и приватными не ограничивает доступ к ним извне. Мы всё ещё можем вызвать метод \_is\_enough из любого места программы:

# Вызываем метод \_is\_enough и спрашиваем его, осталось ли хотя бы 10 мл напитка.

coffee.\_is\_enough(10)

>>> True

Атрибуты и методы, объявленные приватными, вызвать напрямую уже нельзя, но есть обходной путь:

# Создаём класс Drink с приватным атрибутом \_\_volume.

class Drink:

\_\_volume = 200

# Создаём экземпляр класса Drink.

coffee = Drink()

# Используем обходной путь, чтобы обратиться к приватному атрибуту.

coffee.\_Drink\_\_volume

>>> 200

1. **Что такое менеджер контекста, зачем он нужен?**

Менеджер контекста в Python - это объект, определяющий контекст выполнения для блока кода, обычно с помощью оператора with. Менеджер контекста отвечает за установку и удаление любых ресурсов, необходимых блоку кода, таких как открытие и закрытие файла или получение и освобождение блокировки.

Контекстные менеджеры в Python — это удивительный механизм, который позволяет гарантировать корректное управление ресурсами и обеспечивать безопасное выполнение кода.

**with** open("file.txt", "r") **as** file:

content = file.read()

print(content)

Контекстные менеджеры являются важной концепцией в Python, которая обеспечивает удобство, безопасность и эффективность при работе с ресурсами, файлами, базами данных, сетевыми соединениями и другими объектами, требующими специального управления жизненным циклом.

1. **Какие способы конкурентного выполнения программ в Python бывают?**
2. **Что такое GIL и как он работает?**
3. **В каких случаях GIL не работает?**
4. **Что такое сборщик мусора и как он работает?**
5. **Что такое потоки и процессы? Чем отличается поток от процесса?**
6. **Что такое системный вызов fork?**
7. **Какие существуют способы синхронизации процессов и потоков?**
8. **Как передавать информацию из одного процесса в другой?**
9. **Что такое эффект гонок? Как с ним бороться?**
10. **Что такое асинхронность?**
11. **Что такое event loop и как он работает?**
12. **Сколько потоков и процессов работает во время асинхронного выполнения кода?**
13. **Для каких задач стоит использовать потоки, для каких - процессы, а для каких - асинхронность?**

**Git**

1. **Что это такое и зачем он нужен?**

Git — это система контроля версий, которая позволяет нескольким разработчикам одновременно работать над одним продуктом. Это настолько важно и полезно, что сегодня практически все программисты в мире пользуются Git.

1. **Что такое local и remote репозитории?**

Local-находится на личном компьютере, remote- на GitHub, Gitlub(на сервере).

1. **Что такое commit?**

Git commit — это команда для записи индексированных изменений в репозиторий Git. Прежде чем создавать очередной коммит, необходимо проиндексировать файлы в рабочей области с помощью команды git-add. Новый коммит будет включать текущие состояния индексированных файлов плюс последние сохраненные состояния неиндексированных (но отслеживаемых) файлов.

1. **Что такое ветка?**

Ветка (Branch) — это отдельная история изменений (коммитов) в рамках одного репозитория. Git поощряет создание отдельных ветокдля каждой решаемой задачи, благодаря чему и достигается эффективная параллельная работа — каждый разработчик может работать со своей историей и не бояться, что изменения в чужом коде повлияют на его работу.

1. **Что такое pull-request / merge-request?**

Pull request — это заявка на слияние кода из разных веток. В процессе слияния Git создаст коммит и покажет все изменения в файле кода: добавленные до разветвления строки подсветятся зеленым цветом, удаленные — красным. Так каждый из разработчиков и менеджер проекта увидят, что произошло с кодом после совместной работы над коммитом. Перед окончательным слиянием (merge) все разработчики должны просмотреть изменения кода (code review) и принять их.

1. **Что такое merge?**

Слияние веток.

1. **Что такое rebase?**

Слияние веток.

1. **Что такое pull?**

Команда git pull используется для извлечения и загрузки содержимого из удаленного репозитория и немедленного обновления локального репозитория этим содержимым.

1. **Чем rebase отличается от merge?**

Merge слияние происходит “параллельно”, а rebase - “вертикально”. Rebase полезен, когда над кодом ведет работу большая команда, это делает историю проекта более понятной и удобной.

Изображение выглядит как диаграмма, снимок экрана, текст, линия

Автоматически созданное описание

Читая официальное руководство Git, говорится, что “rebase повторно применяет коммиты поверх другой базовой ветви”, тогда как “merge объединяет две или более истории разработки вместе”. Другими словами, ключевое различие между merge и rebase заключается в том, что в то время как merge сохраняет историю в том виде, в каком она произошла, rebase переписывает ее.

1. **Что такое конфликты? Почему они возникают и как их разрешать?**

Обычно конфликты возникают, когда два человека изменяют одни и те же строки в файле или один разработчик удаляет файл, который в это время изменяет другой разработчик. В таких случаях Git не может автоматически определить, какое изменение является правильным. Конфликты затрагивают только того разработчика, который выполняет слияние, остальная часть команды о конфликте не знает. Git помечает файл как конфликтующий и останавливает процесс слияния. В этом случае ответственность за разрешение конфликта несут разработчики.

1. **Уметь пользоваться командами: clone, commit, push, pull, merge, rebase**

**Git clone <ссылка на реп>– клонирование с remote на local.**

**Git commit -m “<>” – индексирование изменений.**

**Git push – отправка/фиксация изменений на remote.**

**Git pull – обновление local данных в ветке.**

**Git merge <ветка, которую сливаем> – слияние веток.**

**Git rebase - слияние веток. (Находимся в ветку, которую хотим слить )**

**Docker**

1. **Что такое контейнеризация? Зачем она нужна?**
2. **Чем контейнер отличается от виртуальной машины?**
3. **Что такое образ?**
4. **Что такое том?**
5. **Что такое сеть контейнера?**
6. **Какие виды сетей бывают?**
7. **Зачем нужен docker-compose?**

**Базы данных**

1. Что такое базы данных и для чего они нужны?
2. Какие типы баз данных бывают?
3. Чем РСУБД отличаются от NoSQL БД?
4. Какие достоинства и недостатки есть у РСУБД и NoSQL?
5. Приведите примеры РСУБД и NoSQL БД.
6. Что такое BASE и ACID?
7. Как понять, какую базу данных нужно использовать в проекте?
8. Что такое индексы? Зачем они нужны? Как они работают?
9. Какие популярные виды индексов существуют?
10. Что такое транзакции? Зачем они нужны?
11. Какие виды изолирования транзакций бывают?
12. Что такое профилирование запросов?

**SQL**

1. Что такое JOIN'ы и какие они бывают?
2. Что такое оконные функции и как они работают?

**Backend**

1. Что такое протокол передачи данных?
2. Что такое HTTP? Чем отличается HTTPS?
3. Какие другие протоколы передачи данных, кроме HTTP(S) вы знаете? Как они работают?
4. Что такое API?
5. Что такое REST и RESTfull API?

**Тестирование**

1. Что такое тестирование и зачем оно нужно?
2. Какие виды тестов вы знаете? Что каждый из видов тестов подразумевает под собой?
3. Какие библиотеки для написания тестов используются?
4. Что такое фикстура в pytest?

**Кодинг:**

1. Написать пример произвольного менеджера контекста.
2. Реализовать паттерн проектирования синглтон.
3. Реализовать паттерн проектирования декоратор.
4. Решить несколько задач.Уметь оценить алгоритмическую сложность решения.

**Задача на временные метки**

Есть список событий:

[

{

"dt": "2022-02-23 04:35:27.353366",

"event": "start"

},

{

"dt": "2022-02-23 04:35:34.654153",

"event": "stop"

},

{

"dt": "2022-02-23 04:38:34.382548",

"event": "start"

},

{

"dt": "2022-02-23 04:38:39.637583",

"event": "stop"

}

]

В этом списке последовательно лежат события типа START и STOP, а также их временные метки. События лежат последовательно: после START всегда идёт STOP и наоборот. Задача:

1. Написать функцию, которая возвращает количество часов, выпадающее на промежуток соответствующих событий START - STOP за указанную дату.
2. Возвращать количество часов за диапазон дат.
3. Возвращать количество часов за текущий день, неделю, месяц, год.