Как проходить собеседования

для Python - разработчиков

Жумабекова Карлыгач

ITcoder

От Автора:

Здравствуй дорогой читатель, эта книга содержит в себе вопросы и ответы на них по языку программирования - **Python**.

Я надеюсь что эта книга поможет тебе трудоустроиться в компанию твоей мечты. В ней есть основные вопросы по **Python**,так же по концепции **ООП** и фреймворку **Django**.

Некоторые вопросы будут повторяться по смыслу но переформулированы. Книга даст тебе первоначальную подготовку к собеседованиям. Но также рекомендуется подготовиться к техническим заданиям самостоятельно. Также советую не слишком надеяться только на эти вопросы ,а все-таки подготовиться по дополнительным вопросам.

Вопросы были перепроверены опытным разработчиком таким как:

А.Эргешов - Python Middle +
Опыт работы: более **4** лет.

А за ответы он не несет ответственности.

Все вопросы к автору.
Личные контакты в Телеграм:
К.Жумабекова - https://t.me/Kjumabekovva

Источники вопросов: Англоязычные сайты и книги.
Источники ответов: Русскоязычные книги по Python, а также ответы от автора.
Вопросы актуальны на 2022 - год.

Q1. Python - интерпретируемый язык или компилируемый**?**

Ответ:

- **PYTHON –** ИНТЕРПРЕТИРУЕМЫЙ ЯЗЫК

Что это означает **–** интерпретируемый**?** А то**,** что код выполняется **(**интерпретируется**)** из исходного текста**,** без предварительного перевода в машинный код**.** Код**,** написанный на компилируемых языках**,** типа **C++,** сначала переводятся в машинный код **(**самый распространенный пример **–** откомпилированного кода **–** файл с расширением **.exe).**

Принято считать**,** что интерпретируемые языки программирования работают медленнее**,** чем компилируемые **–** из**-**за того**,** что трансляция осуществляется не сразу**.** Но отладка и написание кода происходит быстрее**,** потому что не нужно ожидать**,** пока компилятор закончит работать**.**

Q2.Каковы преимущества использования **Python?**

Ответ:

Преимущества использования языка **Python** следующие**.**

- Простота использования **- Python** является языком программирования высокого уровня**,** который легко применять**,** читать**,** писать и изучать**.**
- Интерпретируемый язык **-** поскольку **Python** является интерпретируемым языком**,** он выполняет код построчно и останавливается**,** если в какой**-**либо строке возникает ошибка**.**
- Динамическая типизация **-** разработчик не назначает типы данных переменным во время кодирования**.** Они автоматически присваиваются во время выполнения**.**
- Бесплатность и открытый исходный код **- Python** можно свободно использовать и применять**.** Он является открытым исходным кодом**.**
- Широкая поддержка библиотек **- Python** имеет обширные библиотеки**,** содержащие практически все необходимые функции**.** Кроме того**,** он предоставляет возможность импортировать другие пакеты с помощью менеджера пакетов **Python (pip).**
- Портативность **-** программы на **Python** могут работать на любой платформе без каких**-**либо изменений**.**
- Структуры данных**,** используемые в **Python,** удобны для пользователя**.**
- Он обеспечивает большую функциональность при меньших затратах на разработку кода**.**

Q3.Что такое **pep 8?**

Ответ:

PEP расшифровывается как **Python Enhancement Proposal.** Это набор правил**,** определяющих**,** как форматировать код **Python** для максимальной читабельности**.**

Q4. Какие виды типов данных есть и перечислите типы данных каждого вида**.**

Ответ:

В **Python** существуют изменяемые и неизменяемые типы**.**
К неизменяемым **(immutable)** типам относятся**:** целые числа **(int),** числа с плавающей точкой **(float),** комплексные числа **(complex),** логические переменные **(bool),** кортежи **(tuple),** строки **(str)** и неизменяемые множества **(frozen set).**
К изменяемым **(mutable)** типам относятся**:** списки **(list),** множества **(set),** словари **(dict).**

Q5. Что такое область видимости переменных**?**

твет:

Область видимости или **scope** определяет контекст переменной**,** в рамках которого ее можно использовать**.** В **Python** есть два типа контекста**:** глобальный и локальный**.**

Глобальный контекст

Глобальный контекст подразумевает**,** что переменная является глобальной**,** она определена вне любой из функций и доступна любой функции в программе**.** Например**:**

example1.py > ...

```
name = "Tom"

def say_hi():
    print("Hello", name)

def say_bye():
    print("Good bye", name)

say_hi()
say_bye()
```

Здесь переменная **name** является глобальной и имеет глобальную область видимости**.** И обе определенные здесь функции могут свободно ее использовать**.**

Локальный контекст

В отличие от глобальных переменных локальная переменная определяется внутри функции и доступна только из этой функции, то есть имеет локальную область видимости:

```python
def say_hi():
    name = "Sam"
    surname = "Johnson"
    print("Hello", name, surname)

def say_bye():
    name = "Tom"
    print("Good bye", name)

say_hi()
say_bye()
```

В данном случае в каждой из двух функций определяется локальная переменная **name.** И хотя эти переменные называются одинаково, но тем не менее это две разных переменных, каждая из которых доступна только в рамках своей функции. Также в функции **say_hi()** определена переменная **surname**, которая также является локальной, поэтому в функции **say_bye()** мы ее использовать не сможем.

Скрытие переменных Есть еще один вариант определения переменной, когда локальная переменная скрывают глобальную с тем же именем:

```python
name = "Tom"

def say_hi():
    name = "Bob"  # скрываем значение глобальной переменной
    print("Hello", name)

def say_bye():
    print("Good bye", name)

say_hi()    # Hello Bob
say_bye()   # Good bye Tom
```

Здесь определена глобальная переменная **name.** Однако в функции **say_hi** определена локальная переменная с тем же именем **name.** И если функция **say_bye** использует глобальную переменную, то функция **say_hi** использует локальную переменную, которая скрывает глобальную.

Если же мы хотим изменить в локальной функции глобальную переменную, а не определить локальную, то необходимо использовать ключевое слово **global:**

```python
name = "Tom"

def say_hi():
    global  name
    name = "Bob" # изменяем значение глобальной переменной
    print("Hello", name)

def say_bye():
    print("Good bye", name)

say_hi()    # Hello Bob
say_bye()   # Good bye Bob
```

nonlocal

Выражение **nonlocal** прикрепляет идентификатор к переменной из ближайщего окружающего контекста **(**за исключением глобального контекста**).** Обычно **nonlocal** применяется во вложенных функциях**,** когда надо прикрепить идентификатор за переменной или параметром окружающей внешней функции**.** Рассмотрим ситуацию**,** где это выражение может пригодиться**:**

```python
def outer():  # внешняя функция
    n = 5

    def inner(): # вложенная функция
        print(n)

    inner()     # 5
    print(n)

outer()
```

Здесь вложенная локальная функция **inner()** выводит на консоль значение переменной **n,** которая определена во внешней функции **outer().** Затем в функции **outer()** вызывается внутренняя функция **inner().**

При вызове функции **outer()** здесь мы ожидаемо увидим на консоли два раза число **5.** Однако в данном случае вложенная функция **inner()** просто получает значение**.** Теперь возьмем другую ситуацию**,** когда вложенная функция присваивает значение переменной**:**

example1.py > ...

```python
def outer():  # внешняя функция
    n = 5

    def inner():    # вложенная функция
        n = 25
        print(n)

    inner()     # 25
    print(n)

outer()     # 5
# 25    - inner
# 5     - outer
```

При присвоении значения во вложенной функции**: n = 25** будет создаваться новая переменная **n,** которая скроет переменную **n** из окружающей внешней функции **outer.** В итоге мы получим при выводе два разных числа**.** Чтобы во вложенной функции указать**,** что идентификатор во вложенной функции будет представлять переменную из окружающей функции**,** применяется выражение **nonlocal:**

example1.py > ...

```python
def outer():  # внешняя функция
    n = 5

    def inner():    # вложенная функция
        nonlocal n  # указываем, что n - это переменная из окружающей функции
        n = 25
        print(n)

    inner()     # 25
    print(n)

outer()
```

Q6. Что такое **introspection?**

Ответ:

Многие языки программирования поддерживают интроспекцию, и **Python** не является исключением. В общем, в контексте объектно-ориентированных языков программирования, интроспекция — это способность объекта во время выполнения получить тип, доступные атрибуты и методы, а также другую информацию, необходимую для выполнения дополнительных операций с объектом.

dir()

Первая функция — это функция **dir()**. Она предоставляет список атрибутов и методов, доступных для указанного объекта, который может быть объявленной переменной или функцией.

```
>>> a = [1, 2, 3]
>>> dir(a)
```

type()

Другой часто используемой функцией интроспекции является функция **type()**. Как видно из названия, эта функция возвращает тип объекта, который может быть примитивным типом данных, объектом, классом или модулем. Давайте посмотрим на примеры ниже:

```
>>> type(1.2)
<class 'float'>
```

Q7. Разница между **is** и **==?** В **Python** для сравнения существует два оператора , а именно **is** и **==** :

Ответ:

```python
first = [1, 2, 3]
second = [1, 2, 3]

print(first == second) # True
print(first is second) # False

third = first
print(first is third) # True
```

Но работают они по разному:

- Сравнение через **==** проверяет значения операндов.
- В то время как **is** проверяет указывают ли операнды на один и тот же объект в памяти.

Q8. Что такое пространства имен в **Python?**

Ответ:

Пространство имен в **python -** это имя**,** которое присваивается каждому объекту в **python.** Объектами являются переменные и функции**.** При создании каждого объекта создается его имя вместе с пространством имен **(**адрес внешней функции**,** в которой находится объект**).** Пространства имен хранятся в **python** как словарь**,** где ключ **-** это пространство имен**,** а значение **-** адрес объекта**.** В **python** существует **4** типа пространств имен**.**

- Встроенные пространства имен **-** эти пространства имен содержат все встроенные объекты в **python** и доступны всегда**,** когда запущен **python.**
- Глобальные пространства имен **-** это пространства имен для всех объектов**,** созданных на уровне основной программы**.**
- Охватывающие пространства имен **-** эти пространства имен находятся на более высоком уровне или во внешней функции**.**
- Локальные пространства имен **-** эти пространства имен находятся на локальном уровне или во внутренней функции**.**

Q9. Что такое декораторы в **Python?**

Ответ:

Декораторы используются для добавления некоторых моделей проектирования в функцию без изменения ее структуры**.** Декораторы обычно определяются перед функцией**,** которую они улучшают**.** Чтобы применить декоратор**,** мы сначала определяем функцию**-**декоратор**.** Затем мы пишем функцию**,** к которой она применяется**,** и просто добавляем функцию**-**декоратор над функцией**,** к которой она должна быть применена**.** Для этого мы используем символ **@** перед декоратором

Q16. Что такое **list, dict comprehension ?**

Ответ:

Генератор словарей и списков **-** это еще один краткий способ определения словарей и списков**.**

Пример генератор списка **x=[i for i in range(5)].**

Приведенный выше код создает список**,** как показано ниже**.**

[0,1,2,3,4]

Пример генератор словаря **x=[i : i+2 for i in range(5)]**

Приведенный выше код создает список следующего вида

[0: 2, 1: 3, 2: 4, 3: 5, 4: 6]

Q10. Каковы основные встроенные типы данных в **Python?**

Ответ:

Основные встроенные типы данных в **Python** следующие**.**
Числа **-** они включают целые числа**,** числа с плавающей точкой и комплексные числа**.** например**, 1, 7.9,3+4i.**

- Список **-** Упорядоченная последовательность элементов называется списком**.** Элементы списка могут принадлежать к разным типам данных**.** Например**, [5,'market',2.4].**
- Кортеж **-** это также упорядоченная последовательность элементов**.** В отличие от списков**,** кортежи неизменяемы**,** то есть их нельзя изменить**.** Например**, (3,'tool',1).**
- Строка **-** последовательность символов называется строкой**.** Они объявляются в одинарных или двойных кавычках**.** Например**,** "Сана"**,** 'Она идет на рынок' и т**.**д**.**
- **Set-** Множество **-** это коллекция уникальных элементов**,** которые не упорядочены**.** Например**, {7,6,8}.**
- Словарь **-** словарь хранит значения в парах ключ и значение**,** где каждое значение может быть доступно через ключ**.** Порядок элементов не важен**.** Например**, {1:**'яблоко'**, 2:**'манго**}.**
- Булевы значения **-** Есть два булевых значения**: True** и **False.**

Q11. В чем разница между файлами **.py** и **.pyc?**

Ответ:

Файлы **.py -** это файлы исходного кода **python.** В то время как файлы **.pyc** содержат байткод файлов **python.** Файлы **.pyc** создаются**,** когда код импортируется из другого источника**.** Интерпретатор преобразует исходные файлы **.py** в файлы **.pyc,** что помогает сэкономить время**.**

Q12. Что такое срез в **Python?**

Ответ:

Slicing используется для доступа к частям последовательностей**,** таких как списки**,** кортежи и строки**.** Синтаксис нарезки следующий **-** **[**начало**:**конец**:**шаг**].** Шаг может быть опущен**.** Когда мы пишем **[start:end],** это возвращает все элементы последовательности от начала **(**включительно**)** до элемента **end-1.** Если начальный или конечный элемент имеет отрицательное значение **i,** то это означает**,** что это **ith** элемент от конца**.** Шаг указывает на прыжок или на то**,** сколько элементов должно быть пропущено**.** Например**,** если есть список **[1,2,3,4,5,6,7,8].** Тогда **[-1:2:2]** будет возвращать элементы**,** начиная с последнего элемента до третьего**,** печатая каждый второй элемент**,** т**.**е**.** **[8,6,4].**

Q13. В чем разница между **list** и **tuples** в **Python?**

List	Tuples
Списки являются изменяемыми, т.е. их можно редактировать.	Кортежи неизменяемы (кортежи - это списки, которые нельзя редактировать).
Списки обрабатываются медленнее, чем кортежи.	Кортежи быстрее, чем списки.
Syntax: **list_1 = [10, 'ITCODER', 20]**	**Syntax:** **tup_1 = (10, 'ITCBOOTCAMP' , 20)**

Q14. Каковы ключевые особенности **Python?**

Ответ:

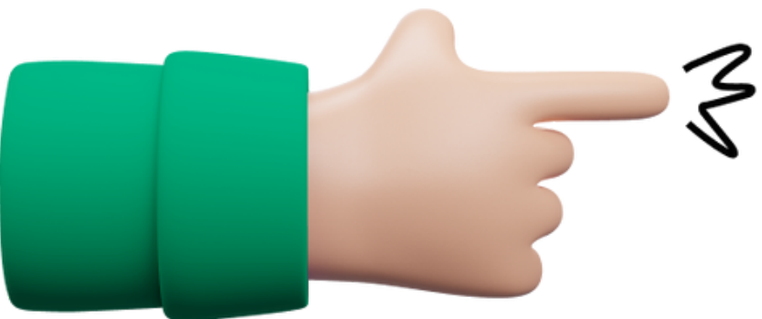
- **Python** является интерпретируемым языком. Это означает, что, в отличие от таких языков, как **C** и его разновидностей, **Python** не нужно компилировать перед запуском. К другим интерпретируемым языкам относятся **PHP** и **Ruby**.
- **Python** динамически типизирован, это означает, что вам не нужно указывать типы переменных при их объявлении или что-то в этом роде. Вы можете делать такие вещи, как **x=111**, а затем **x=**"Я строка" без ошибок.
- **Python** хорошо подходит для объектно-ориентированного программирования, поскольку он позволяет определять классы, а также компоновать и наследовать их. В **Python** нет указателей доступа (как в **C++ - public**, **private**).
- В **Python** функции являются объектами первого класса. Это означает, что их можно присваивать переменным, возвращать из других функций и передавать в функции. Классы также являются объектами первого класса.
- Код на **Python** пишется быстро, но выполняется он зачастую медленнее, чем на компилируемых языках. К счастью, **Python** позволяет включать расширения, основанные на языке **C**, поэтому узкие места могут быть оптимизированы, что часто и происходит. Хорошим примером этого является пакет **numpy**, который действительно работает довольно быстро, потому что большая часть вычислений, которые он выполняет, на самом деле выполняется не в **Python**.
- **Python** находит применение во многих сферах - веб-приложения, автоматизация, научное моделирование, приложения для работы с большими данными и многое другое. Он также часто используется в качестве "клеящего" кода, чтобы заставить другие языки и компоненты хорошо работать.

Q15. Что такое ключевые слова в **Python?**

Ответ:

Ключевые слова в **python -** это зарезервированные слова**,** которые имеют особое значение**.** Они обычно используются для определения типа переменных**.** Ключевые слова нельзя использовать для имен переменных или функций**.** В **python** есть следующие **33** ключевых слова**-**

- **And**
- **Or**
- **Not**
- **If**
- **Elif**
- **Else**
- **For**
- **While**
- **Break**
- **As**
- **Def**
- **Lambda**
- **Pass**
- **Return**
- **True**
- **False**
- **Try**
- **With**
- **Assert**
- **Class**
- **Continue**
- **Del**
- **Except**
- **Finally**
- **From**
- **Global**
- **Import**
- **In**
- **Is**
- **None**
- **Nonlocal**
- **Raise**
- **Yield**



Q16.Что такое литералы в **Python** и объясните различные литералы**.**

Ответ:

Литерал в исходном коде **python** представляет собой фиксированное значение для примитивных типов данных**.** В **python** существует **5** типов литералов**.**

1. Строковые литералы **-** строковый литерал создается путем присвоения переменной некоторого текста**,** заключенного в одинарные или двойные кавычки**.** Чтобы создать многострочный литерал**,** присвойте многострочный текст**,** заключенный в тройные кавычки**.** Например**, name="Makers".**
2. Символьный литерал **-** создается путем присвоения переменной одного символа**,** заключенного в двойные кавычки**.** Например**, a='t'**
3. Числовые литералы **-** они включают числовые значения**,** которые могут быть целым числом**,** значением с плавающей точкой или комплексным числом**.** Например**, a=50**
4. Булевы литералы **-** они могут иметь два значения **- True** или **False.**
5. Коллекции литералов **-** они бывают четырех типов**:**

    a) коллекции списков **-** например**, a=[1,2,3,'Anas'].**

    b) Литералы кортежей **-** например**, a=(5,6,7,8)**

    c) Словарные литералы **-** Например**, dict={1: 'apple', 2: 'mango, 3: 'banana'}**

    d) Литералы множества **-** Например**, {"Anas", "Yrys",** "M**aks"}.**

6. Специальный литерал**-** В **Python** есть один специальный литерал **None,** который используется для возврата нулевой переменной**.**

Q17. Как объединить массивы данных в **pandas?**

Ответ:

Массивы данных в **python** можно объединить следующими способами**.**

- Конкатенировать их**,** складывая **2** массива данных вертикально**.**
- Конкатенация путем укладки двух массивов данных по горизонтали**.**
- Объединение их на общем столбце**.** Это называется объединением**.**
- Для объединения двух массивов данных используется функция **concat().** Ее синтаксис таков **- pd.concat([dataframe1, dataframe2]).**
- Массивы данных объединяются по общему столбцу**,** называемому ключом**.** Когда мы объединяем все строки в массиве данных**,** это объединение**,** а используемое соединение **-** внешнее соединение**.** Если же мы объединяем общие строки или пересечения**,** то используется внутреннее объединение**.** Его синтаксис таков **- pd.concat([dataframe1, dataframe2], axis='axis', join='type_of_join)**

Q18. Какие новые возможности добавлены в версии **Python 3.9.0.0?**

Ответ:

Новые возможности в версии **Python 3.9.0.0** следующие**.**

- Новые словарные функции **Merge(|)** и **Update(|=)**
- Новые строковые методы для удаления префиксов и суффиксов
- Генераторы с подсказкой типов в стандартных коллекциях
- Новый парсер**,** основанный на **PEG,** а не на **LL1**
- Новые модули**,** такие как **zoneinfo** и **graphlib**
- Улучшенные модули**,** такие как **ast, asyncio** и т.д**.**
- Оптимизации**,** такие как оптимизированная идиома для присваивания**,** обработка сигналов**,** оптимизированные встроенные средства **python** и т.д**.**
- Устранение устаревших функций и команд**,** таких как устаревшие модули парсера и символов**,** устаревшие функции и т.д**.**
- Удаление ошибочных методов**,** функций и т.д**.**

Q19. Как осуществляется управление памятью в **Python?**

Ответ:

Управление памятью в **Python** осуществляется следующими способами**:**

- Управление памятью в **Python** осуществляется с помощью частного пространства памяти **Python.** Все объекты и структуры данных **Python** располагаются в приватной памяти**.** Программист не имеет доступа к этой приватной памяти**.** Вместо него об этом заботится интерпретатор **Python.**
- Распределение пространства памяти для объектов **Python** осуществляется менеджером памяти **Python. API** ядра предоставляет программисту доступ к некоторым инструментам для написания кода**.**
- **Python** также имеет встроенный сборщик мусора**,** который перерабатывает всю неиспользуемую память**,** чтобы она могла быть доступна для пространства динамической памяти**.**

Q20. Что такое пространство имен в **Python?**

Ответ:

Пространство имен **-** это система именования**,** используемая для обеспечения уникальности имен во избежание конфликтов именования**.**

Q21. Что такое **PYTHONPATH?**

Ответ:

Это переменная окружения, которая используется при импорте модуля. Всякий раз, когда модуль импортируется, **PYTHONPATH** также просматривается, чтобы проверить наличие импортированных модулей в различных каталогах. Интерпретатор использует ее, чтобы определить, какой модуль загрузить.

Q22. Что такое модули **Python?** Назовите некоторые часто используемые встроенные модули в **Python?**

Ответ:

Модули **Python -** это файлы, содержащие код **Python.** Этот код может быть либо функциями классами, либо переменными. Модуль **Python -** это файл **.py,** содержащий исполняемый код.

Вот некоторые из часто используемых встроенных модулей:

- **os**
- **sys**
- **math**
- **random**
- **data time**
- **JSON**

Q23.Что такое лямбда-функция**?**

Ответ:

Анонимная функция известна как **lambda** функция. Эта функция может иметь любое количество параметров, но может иметь только один оператор.

Пример:

- **a = lambda x,y : x+y**
- **print(a(5, 6))**

Вывод: **11**

Q24. Чувствителен ли **python** к регистру**?**

Ответ:

Да**. Python -** это язык**,** чувствительный к регистру**.**

Q25: Что такое преобразование типов в **Python?**

Ответ:

Преобразование типов **-** это преобразование одного типа данных в другой**.**

int() - преобразование любого типа данных в целое число

float() - преобразует любой тип данных в тип **float**

ord() - преобразует символы в целое число

hex() - преобразует целые числа в шестнадцатеричные

oct() - преобразует целое число в восьмеричное

tuple() - Эта функция используется для преобразования в кортеж**.**

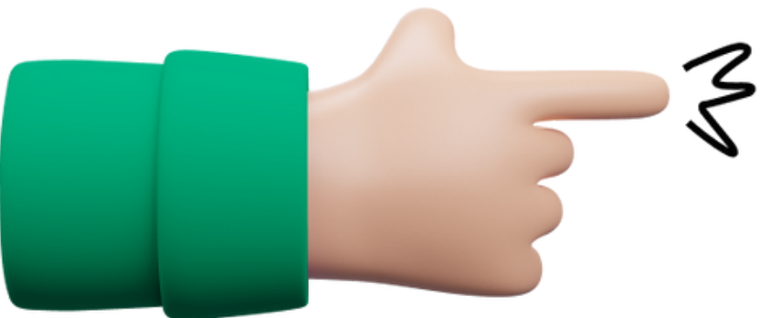
set() - Эта функция используется для преобразования в множество**.**

list() - Эта функция используется для преобразования любого типа данных в тип списка**.**

dict() - Эта функция используется для преобразования кортежа порядка **(**ключ**,** значение**)** в словарь**.**

str() - Используется для преобразования целого числа в строку**.**

complex(real,imag) - Эта функция преобразует вещественные числа в комплексные **(real,imag).**



Q26. Нужны ли отступы в **Python?**

Ответ:

Отступы необходимы в **Python.** Он определяет блок кода. Весь код в циклах, классах, функциях и т.д. указывается в блоке с отступом. Обычно это делается с помощью четырех символов пробела. Если ваш код не имеет отступов, он не будет выполняться точно, а также будет выдавать ошибки.

Q27. В чем разница между массивами и списками **Python?**

Ответ:

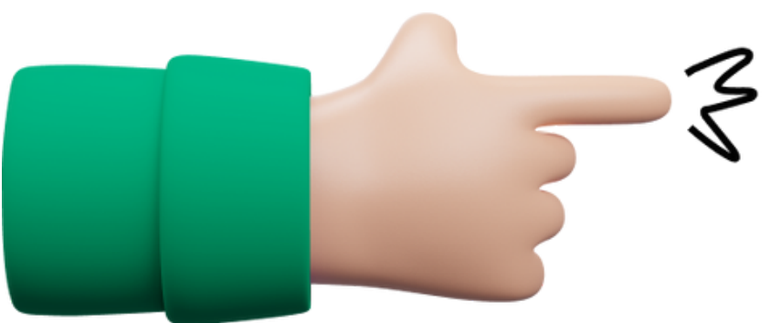
Массивы и списки в **Python** имеют одинаковый способ хранения данных. Но массивы могут содержать только элементы одного типа данных, в то время как списки могут содержать элементы любого типа данных.

Пример:

```
1. import array as arr
2. My_Array=arr.array('i',[1,2,3,4])
3. My_list=[1,'abc',1.20]
4. print(My_Array)
5. print(My_list)
```

Вывод:

```
array('i', [1, 2, 3, 4])
[1, 'abc', 1.2]
```

Q28. Что такое функции в **Python?**

Ответ:

Функция **-** это блок кода**,** который выполняется только тогда**,** когда его вызывают**.** Для определения функции в **Python** используется ключевое слово **def.**

Пример**:**

```
1. def Itcoder():
2.         print("Привет, добро пожаловать в Itcoder")
3. Itcoder(); #вызов функции
```

Вывод**:** Привет**,** добро пожаловать в **Itcoder**

Q29.Что такое **__init__?**

Ответ:

__init__ - это метод или конструктор в **Python.** Этот метод автоматически вызывается для выделения памяти при создании нового объекта**/** экземпляра класса**.** Все классы имеют метод **__init__.**

Ниже приведен пример его использования**.**

```
class Employee:
        def __init__(self, name, age, salary):
                self.name = name
                self.age = возраст
                self.salary = 20000
E1 = Employee("XYZ", 23, 20000)
# E1 - это экземпляр класса Employee.
#__init__ выделяет память для E1.
print(E1.name)
print(E1.age)
print(E1.salary)
```

вывод**:**

```
XYZ
23
20000
```

Q30. Что такое **self** в **Python?**

Ответ:

Self - это экземпляр или объект класса**.** В **Python** это явно включено в качестве первого параметра**.** Однако это не так в **Java,** где он необязателен**.** Это помогает различать методы и атрибуты класса с локальными переменными**.**

Переменная **self** в методе **init** относится к вновь созданному объекту**,** в то время как в других методах она относится к объекту**,** метод которого был вызван**.**

Q31. Как работают **break, continue** и **pass?**

Ответ:

break	*Позволяет завершить цикл при выполнении некоторого условия и передать управление на следующий оператор.*
continue	*Позволяет пропустить часть цикла при выполнении некоторого условия и передаче управления в начало цикла.*
pass	*Используется, когда вам нужен некоторый блок кода синтаксически, но вы хотите пропустить его выполнение. По сути, это нулевая операция. При ее выполнении ничего не происходит.*

Q32. Что делает **[::-1]?**

Ответ:

[::-1] используется для изменения порядка массива или последовательности**.**

Например**:**

1. **My_Array=['i',[1,2,3,4,5]]**
2. **My_Array[::-1]**

Выходные данные**: ['i', [5, 4, 3, 2, 1]]**

[::-1] перепечатывает обратную копию упорядоченных структур данных**,** таких как массив или список**.** исходный массив или список остается неизменным**.**

Q33. Как в **Python** можно произвольно распределить элементы списка по местам**?**

Ответ:

Рассмотрим пример**,** показанный ниже**:**

```
1. from random import shuffle
2. x = ['Keep', 'The', 'Blue', 'Flag', 'Flying', 'High'].
3. shuffle(x)
4. print(x)
```

Вывод следующего кода выглядит следующим образом**.**

['Flying', 'Keep', 'Blue', 'High', 'The', 'Flag'].

Q34. Что такое итераторы в **python?**

Ответ:

Итераторы **-** это объекты**,** которые можно обходить или итерировать**.**

Q35. Как можно генерировать случайные числа в **Python?**

Ответ:

Модуль **Random -** это стандартный модуль**,** который используется для генерации случайного числа**.** Метод определяется следующим образом**:**

```
1. import random
2. random.random
```

1. Метод **random.random()** возвращает число с плавающей точкой**,** находящееся в диапазоне **[0, 1).** Функция генерирует случайные числа с плавающей точкой**.** Методы**,** которые используются с классом **random,** являются связанными методами скрытых экземпляров**.** Экземпляры **Random** могут быть использованы для демонстрации многопоточных программ**,** которые создают различные экземпляры для отдельных потоков**.** Другие генераторы случайных чисел**,** которые используются в данном случае**,** следующие**:**

- **randrange(a, b):** выбирается целое число и определяется диапазон между **[a, b).** Он возвращает элементы**,** выбирая их случайным образом из указанного диапазона**.** Объект диапазона не формируется**.**
- **uniform(a, b):** выбирает число с плавающей точкой**,** которое определено в диапазоне **[a,b].** Возвращает число с плавающей точкой**.**
- **normalvariate(mean, sdev):** используется для получения нормального распределения**,** где **mean -** это среднее значение**,** а **sdev -** сигма**,** которая используется для стандартного отклонения**.**
- Используемый и внедряемый класс **Random** создает независимые множественные генераторы случайных чисел**.**

Q36. В чем разница между **range** и **xrange?**

Ответ:

По большому счету**, xrange** и **range** совершенно одинаковы по функциональности**.** Они оба предоставляют способ создания списка целых чисел для использования по своему усмотрению**.** Единственное различие заключается в том**,** что **range** возвращает объект списка **Python,** а **x range** возвращает объект **xrange.**

Это означает**,** что **xrange** не генерирует статический список во время выполнения**,** как это делает **range.** Он создает значения по мере необходимости с помощью специальной техники**,** называемой **yielding.** Эта техника используется с типом объектов**,** известным как генераторы**.** Это означает**,** что если у вас есть действительно гигантский диапазон**,** для которого вы хотели бы сгенерировать список**,** скажем**,** на один миллиард**,** следует использовать функцию **xrange.**

Это особенно актуально**,** если у вас очень чувствительная к памяти система**,** например**,** мобильный телефон**,** с которым вы работаете**,** поскольку **range** будет использовать столько памяти**,** сколько сможет**,** чтобы создать ваш массив целых чисел**,** что может привести к ошибке памяти и краху вашей программы**.** Это зверь**,** жаждущий памяти**.**

Q37. Как писать комментарии в **python?**

Ответ:

Комментарии в **Python** начинаются с символа **#.** Однако иногда в качестве альтернативы для комментариев используются **docstrings (**строки**,** заключенные в тройные кавычки**).**

Пример**:**

```
1. age=22 #Комментарии в Python начинаются следующим образом
2. print(" Комментарии в Python начинаются с #")
```

Вывод**:** Комментарии в **Python** начинаются с символа **#**

Q38. Что такое **pickling** и **unpickling?**

Ответ:

Модуль **Pickle** принимает любой объект **Python,** преобразует его в строковое представление и сбрасывает его в файл с помощью функции **dump,** этот процесс называется **pickling.** А процесс извлечения исходных объектов **Python** из сохраненного строкового представления называется **unpickling.**

Q39. Что такое генераторы в **python?**

Ответ:

Функции**,** которые возвращают итерируемый набор элементов**,** называются генераторами**.**

Q40. Как набрать первую букву строки заглавными буквами**?**

Ответ:

В **Python** метод **capitalize()** выводит первую букву строки заглавной**.** Если строка уже содержит заглавную букву в начале**,** то возвращается исходная строка**.**

Q41. Как преобразовать строку в нижний регистр**?**

Ответ:

Для преобразования строки в нижний регистр можно использовать функцию **lower().**

Пример**:**

```
1. stg='MAKERS'
2. print(stg.lower())
```

Выходные данные**: makers**

Q42. Как закомментировать несколько строк в **python?**

Ответ:

Многострочные комментарии появляются более чем в одной строке**.** Все комментируемые строки должны иметь префикс **#.** Вы также можете использовать очень хороший метод быстрого комментирования нескольких строк**.** Все**,** что вам нужно сделать**,** это удерживать клавишу **ctrl** и щелкнуть левой кнопкой мыши в каждом месте**,** где вы хотите включить символ **#,** и набрать **#** только один раз**.** В результате будут закомментированы все строки**,** на которых вы установили курсор**.** Или же выделите некую часть кода и нажмите **ctrl+/**

Q 43.Что такое **docstrings** в **Python?**

Ответ:

docstrings - это не комментарии**,** а строки документации**.** Эти строки заключаются в тройные кавычки**.** Они не присваиваются никакой переменной и поэтому иногда служат в качестве комментариев**.**

Пример**:**

```
1. """
2.       Использование docstring в качестве комментария.
3.       Этот код делит 2 числа
4. """
5. x=8
6. y=4
7. z=x/y
8. print(z)
```

Вывод**: 2.0**

Q44. Каково предназначение операторов **'is', 'not'** и **'in'?**

Ответ:

Операторы **-** это специальные функции**.** Они принимают одно или несколько значений и выдают соответствующий результат**.**

- **is:** возвращает истину**,** если **2** операнда истинны **(**пример**: "a" - is 'a').**
- **not:** возвращает обратную величину булевого значения**.**
- **in:** проверяет**,** присутствует ли некоторый элемент в некоторой последовательности**.**

Q45. Как используются функции **help()** и **dir()** в **Python?**

Ответ:

Функции **help()** и **dir()** доступны из интерпретатора **Python** и используются для просмотра общего списка встроенных функций**.**
Функция **help():** Функция **help()** используется для отображения строки документации**,** а также облегчает просмотр справки**,** связанной с модулями**,** ключевыми словами**,** атрибутами и т**.**д**.**
Функция **Dir():** Функция **dir()** используется для отображения заданных символов**.**

Q46. Почему при выходе из **Python** не происходит удаление всей памяти**?**

Ответ:

1. При выходе из **Python,** особенно в тех модулях **Python,** которые имеют круговые ссылки на другие объекты или объекты, на которые ссылаются из глобальных пространств имен, не всегда происходит удаление или освобождение памяти**.**
2. Невозможно удалить те участки памяти, которые зарезервированы библиотекой **C.**
3. При выходе, поскольку **Python** имеет свой собственный эффективный механизм очистки, он попытается освободить**/**уничтожить все остальные объекты**.**

Q47. Что такое словарь в **Python?**

Ответ:

Встроенный тип данных в **Python** называется словарем**.** Он определяет отношения один**-**к**-**одному между ключами и значениями**.** Словари содержат пару ключей и соответствующих им значений**.** Словари индексируются по ключам**.**

Рассмотрим пример**:**

Следующий пример содержит несколько ключей**.** Страна**,** Столица и Национальность**.** Их соответствующие значения **-** Кыргызстан**,** Бишкек и Кыргыз соответственно**.**

```
1. dict1={'Country':'Kyrgyzstan','Capital':'Bishkek','Nation':'Kyrgyz'}
2. print dict1['Country']
```

Вывод**:Kyrgyzstan**

```
1. print dict1['Capital']
```

Вывод**:Bishkek**

```
1. print dict['Nation']
```

Вывод**:Kyrgyz**

Q48. Как можно использовать тройные операторы в **python?**

Ответ:

Троичные операторы **-** это операторы**,** которые используются для отображения условных операторов**.** Он состоит из значений **true** или **false** с утверждением**,** которое должно быть проверено**.**

Синтаксис**:**

Троичный оператор будет представлен в виде**:**

[on_true] if [выражение**] else [on_false]x, y = 25, 50big = x if x < y else y**

Пример**:**

Выражение оценивается как **if x<y else y**, в этом случае**,** если **x<y** истинно**,** то значение возвращается как **big=x,** а если оно неверно**,** то в результате будет отправлено **big=y.**

Q49. Что означает**: *args, **kwargs?** И зачем мы их используем**?**

Ответ:

Мы используем ***args,** когда не уверены**,** сколько аргументов будет передано в функцию**,** или если хотим передать в функцию сохраненный список или кортеж аргументов**. **kwargs** используется**,** когда мы не знаем**,** сколько аргументов ключевых выражений будет передано функции**,** или может использоваться для передачи значений словаря в качестве аргументов ключевых выражений**.** Идентификаторы **args** и **kwargs -** это соглашение**,** вы также могли бы использовать ***bob** и ****billy,** но это было бы неразумно**.**

Q50. Что делает функция **len()?**

Ответ:

Она используется для определения длины строки**,** списка**,** массива и т**.**д**.**

Например**:**

```
1. itc='ITCODER'
2. len(itc)
```

Вывод**: 7**

Q51. Объясните методы **split(), sub(), subn()** модуля **"re"** в **Python.**

Ответ:

Для модификации строк модуль **"re"** в **Python** предоставляет **3** метода**.** К ним относятся**:**

1. **split() -** использует шаблон **regex** для "разбиения" заданной строки на список**.**
2. **sub() -** находит все подстроки**,** в которых совпадает **regex-**шаблон**,** и заменяет их другой строкой**.**
3. **subn() -** аналогична **sub()** и также возвращает новую строку вместе с количеством замен

Q52. Что такое отрицательные индексы и почему они используются**?**

Ответ:

Последовательности в **Python** индексируются и состоят как из положительных**,** так и отрицательных чисел**.** Положительные числа используют **'0'** в качестве первого индекса и **'1'** в качестве второго индекса**,** и процесс продолжается в том же духе**.**

Индекс для отрицательных чисел начинается с **'-1',** который представляет собой последний индекс в последовательности**,** и **'-2'** в качестве предпоследнего индекса**,** и последовательность продолжается так же**,** как и для положительных чисел**.**

Отрицательный индекс используется для удаления из строки пробелов новой строки и позволяет строке исключить последний символ**,** который задается как **S[:-1].** Отрицательный индекс также используется для того**,** чтобы показать индекс для представления строки в правильном порядке

Q53. Что такое пакеты **Python?**

Ответ:

Пакеты **Python -** это пространства имен**,** содержащие несколько модулей**.**

Вопрос **54:** Как можно удалять файлы в **Python?**

Ответ:

Чтобы удалить файл в **Python,** необходимо импортировать модуль **OS.**
После этого нужно использовать функцию **os.remove().**
Пример**:**

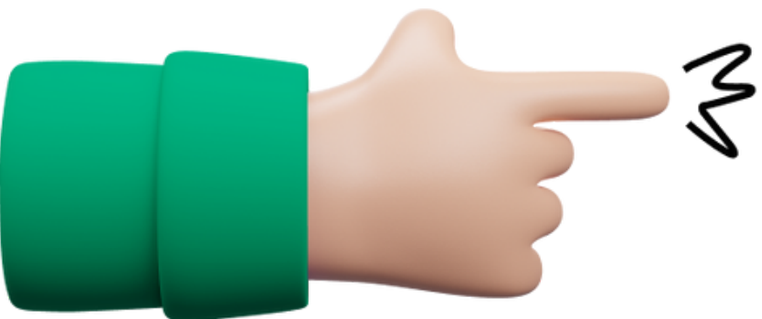
```
1. import os
2. os.remove("itc.txt")
```

Q54. Какие встроенные типы есть в **python?**

Ответ:

Встроенные типы в **Python** следующие **:**

- Целые числа
- С плавающей точкой
- Комплексные числа
- Строки
- Булевы
- Встроенные функции



Q55. Какие преимущества имеют массивы **NumPy** перед (вложенными) списками **Python?**

Ответ:

Списки **Python -** это эффективные контейнеры общего назначения. Они поддерживают (достаточно) эффективную вставку, удаление, добавление и конкатенацию, а возможности понимания списков в **Python** позволяют легко конструировать и манипулировать ими.

У них есть определенные ограничения: они не поддерживают "векторные" операции, такие как поэлементное сложение и умножение, а тот факт, что они могут содержать объекты разных типов, означает, что **Python** должен хранить информацию о типе для каждого элемента и выполнять код распределения типов при работе с каждым элементом. **NumPy** не только более эффективен, но и более удобен. Вы получаете множество векторных и матричных операций бесплатно, что иногда позволяет избежать лишней работы. К тому же они эффективно реализованы.

Массив **NumPy** быстрее, и вы получаете много встроенных в **NumPy** функций, **FFT**, преобразования, быстрый поиск, базовую статистику, линейную алгебру, гистограммы и т.д.

Q56. Как добавить значения в массив в **python?**

Ответ:

Элементы можно добавить в массив с помощью функций **append(), extend()** и **insert (i,x).**

Пример:

```
1. a=['d', [1.1 , 2.1 , 3.1] ]
2. a.append(3.4)
3. print(a)
4. a.extend([4.5,6.3,6.8])
5. print(a)
6. a.insert(2,3.8)
7. print(a)
```

Вывод:

['d', [1.1, 2.1, 3.1, 3.4]]

['d', [1.1, 2.1, 3.1, 3.4, 4.5, 6.3, 6.8]]

['d', [1.1, 2.1, 3.8, 3.1, 3.4, 4.5, 6.3, 6.8]]

Q57. Как удалить значения из массива в **python?**

Ответ:

Элементы массива можно удалить с помощью метода **pop()** или **remove().** Разница между этими двумя функциями заключается в том**,** что первая возвращает удаленное значение**,** а вторая **-** нет**.**

Пример**:**

```
1. a=['d', [1.1, 2.2, 3.8, 3.1, 3.7, 1.2, 4.6]]
2. print(a.pop())
3. print(a.pop(3))
4. a.remove(1.1)
5. print(a)
```

Вывод**:**

4.6

3.1

[‘d’, [2.2, 3.8, 3.7, 1.2]]

Q58. Есть ли в **Python** концепции ООП**?**

Ответ:

Python является объектно**-**ориентированным языком программирования**.** Это означает**,** что любая программа может быть решена в **python** путем создания объектной модели**.** Однако **Python** можно рассматривать как процедурный**,** так и структурный язык**.**

Q59. Как осуществляется многопоточность в **Python?**

Ответ:

В **Python** есть пакет многопоточности, но если вы хотите использовать многопоточность для ускорения кода, то обычно это не очень хорошая идея.

В **Python** есть конструкция под названием **Global Interpreter Lock (GIL).** **GIL** гарантирует, что только один из ваших "потоков" может выполняться в любой момент времени. Поток получает **GIL,** выполняет небольшую работу, а затем передает **GIL** следующему потоку.

Это происходит очень быстро, поэтому для человеческого глаза может показаться, что ваши потоки выполняются параллельно, но на самом деле они просто по очереди используют одно и то же ядро процессора.

Вся эта передача **GIL** добавляет лишние затраты на выполнение. Это означает, что если вы хотите, чтобы ваш код выполнялся быстрее, то частое использование пакета потоков не является хорошей идеей.

Q60. Что представляет собой процесс компиляции и компоновки в **python?**

Ответ:

Компиляция и компоновка позволяют правильно скомпилировать новые расширения без каких-либо ошибок, а компоновка может быть выполнена только тогда, когда они проходят процедуру компиляции.

Если используется динамическая загрузка, то это зависит от стиля, который поставляется с системой. Интерпретатор **python** может быть использован для обеспечения динамической загрузки файлов настройки конфигурации и перестроит интерпретатор.

Для этого необходимо выполнить следующие шаги:

- Создайте файл с любым именем и на любом языке, который поддерживается компилятором вашей системы. Например, **file.c** или **file.cpp.**
- Поместите этот файл в каталог **Modules/** дистрибутива, который используется.
- Добавьте строку в файл **Setup.local**, который присутствует в каталоге **Modules/.**
- Запустите файл с помощью спама **file.o**
- После успешного запуска пересоберите интерпретатор с помощью команды **make** в каталоге верхнего уровня.
- Если файл был изменен, запустите команду **rebuildMakefile,** используя команду **'make Makefile'.**

Q61. Что такое метакласс, переменная цикла?

Ответ:

Метаклассы – это классы, экземпляры которых являются классами. Чтобы создать свой собственный метакласс в **Python**, нужно воспользоваться подклассом **type**, стандартным метаклассом в **Python**. Чаще всего метаклассы используются в роли виртуального конструктора. Чтобы создать экземпляр класса, нужно сначала вызвать этот самый класс. Точно так же делает и **Python:** для создания нового класса вызывает метакласс. Метаклассы определяются с помощью базовых классов в атрибуте **__metaclass__**. При создании класса допускается использование методов **__init__** и **__new__**. С их помощью можно пользоваться дополнительными функциями. Во время выполнения оператора **class** генерируется пространство имен, которое будет содержать атрибуты будущего класса. Затем, для непосредственного создания, вызывается метакласс с именем и атрибутами.

переменная цикла:

Вот простейший пример использования цикла, где в качестве множества значений используется кортеж:

```
i = 1
for color in 'red', 'orange', 'yellow', 'green', 'cyan', 'blue', 'violet':
    print(i, '-th color of rainbow is ', color, sep = '')
    i += 1
```

В этом примере переменная color последовательно принимает значения ‘red’, ‘orange’ и т.д. В теле цикла выводится сообщение, которое содержит название цвета, то есть значение переменной color, а также номер итерации цикла – число, которое сначала равно 1, а потом увеличивается на один (инструкцией

```
i += 1
```

с каждым проходом цикла).

Q62.Объясните разницу между списком и кортежем?

Ответ:

Список изменяемый, а кортеж — нет. Кортежи можно хешировать, как в случае создания ключей для словарей.

Q63. В чем разница между потоками и процессами**?**

Ответ:

Для начала нам нужно разобраться с терминами**:**

- процесс **(process) -** это**,** грубо говоря**,** запущенная программа**.** Процессу выделяются отдельные ресурсы**:** память**,** процессорное время
- поток **(thread) -** это единица исполнения в процессе**.** Потоки разделяют ресурсы процесса**,** к которому они относятся**.**

Python оптимизирован для работы в однопоточном режиме**.** Это хорошо**,** если в программе используется только один поток**.** И**,** в то же время**,** у **Python** есть определенные нюансы работы в многопоточном режиме**.** Связаны они с тем**,** что **CPython** использует **GIL (global interpreter lock).**

GIL не дает нескольким потокам исполнять одновременно код **Python.** Если не вдаваться в подробности**,** то **GIL** можно представить как некий переходящий флаг**,** который разрешает потокам выполняться**.** У кого флаг**,** тот может выполнять работу**.** Флаг передается либо каждые сколько**-**то инструкций **Python**, либо**,** например**,** когда выполняются какие**-**то операции ввода**-**вывода**.**

Поэтому получается**,** что разные потоки не будут выполняться параллельно**,** а программа просто будет между ними переключаться**,** выполняя их в разное время**.** Однако**,** если в программе есть некое **«**ожидание**»:** пакетов из сети**,** запроса пользователя**,** пауза типа **time.sleep**, то в такой программе потоки будут выполняться как будто параллельно**.** А всё потому**,** что во время таких пауз флаг **(GIL)** можно передать другому потоку**.**

То есть**,** потоки отлично подходят для задач**,** которые связаны с операциями ввода**-**вывода**:**

- Подключение к оборудованию и подключение по сети в целом
- Работа с файловой системой
- Скачивание файлов по сети

Процессы

Процессы позволяют выполнять задачи на разных ядрах компьютера. Это важно для задач, которые завязаны на **CPU**. Для каждого процесса создается своя копия ресурсов, выделяется память, у каждого процесса свой **GIL**. Это же делает процессы более тяжеловесными, по сравнению с потоками.

Кроме того, количество процессов, которые запускаются параллельно, зависит от количества ядер и **CPU** и обычно исчисляется в десятках, тогда как количество потоков для операций ввода-вывода может исчисляться в сотнях.

Процессы и потоки можно совмещать, но это усложняет программу и на базовом уровне для операций ввода-вывода лучше остановиться на потоках.

Примечание

Совмещение потоков и процессов, то есть запуск процесса в программе и внутри него уже запуск потоков - сильно усложняет траблшутинг программы. И лучше такой вариант не использовать.

Несмотря на то, что, как правило, для задач ввода-вывода лучше использовать потоки с некоторыми модулями надо использовать процессы, так как они могут некорректно работать с потоками.

Примечание

Помимо процессов и потоков есть еще один вариант одновременного подключения к оборудованию: асинхронное программирование.

подробнее:

https://medium.com/nuances-of-programming/потоковые-и-многопроцессорные-модули-на-**python-1a86a6d8986f**

Q64.Разница между генераторами и итераторами**?**

Ответ:

В **Python** итераторы используются для перебора группы элементов **(**например**,** в списке**).** Генераторы представляют собой способ реализации итераторов**.** В них применяется **yield** для возврата выражения из функции**,** но в остальном генератор ведет себя как обычная функция**.**

Q65. Какие есть виды импорта**?**

Есть 4 разных вида импортов:

1. `import <пакет>`
2. `import <модуль>`
3. `from <пакет> import <модуль или подпакет или объект>`
4. `from <модуль> import <объект>`

Q66. Что такое класс**,** итератор**,** генератор**?**

Ответ:

Класс в **Python —** это логическая группа данных и функций**.** Он дает возможность создавать структуры данных**,** которые содержат произвольный контент и**,** следовательно**,** легко доступны**.**

```
class myClass():
    def method1(self):
        print "Guru99"

    def method2(self,someString):
        print "Software Testing:" + someString

```

Итератор **(iterator) -** это объект**,** который возвращает свои элементы по одному за раз**.** С точки зрения **Python -** это любой объект**,** у которого есть метод **__next__ .** Этот метод возвращает следующий элемент**,** если он есть**,** или возвращает исключение **StopIteration,** когда элементы закончились**.**

Генератор в **Python —** это функция с уникальными возможностями**.** Она позволяет приостановить или продолжить работу**.** Генератор возвращает итератор**,** по которому можно проходить пошагово**,** получая доступ к одному значению с каждой итерацией**.**

подробнее:
https://pythonru.com/uroki/30-generatory-dlja-nachinajushhih

Q67. Что такое библиотеки **Python?** Назовите несколько из них**.**

Ответ:

Библиотеки **Python -** это коллекция пакетов **Python.** Некоторые из наиболее часто используемых библиотек **Python - Numpy, Pandas, Matplotlib, Scikit-learn** и многие другие**.**

Q68. Для чего используется метод **split?**

Ответ:

Метод **split()** используется для разделения заданной строки в **Python.**
Пример**:**
a="itcoder python"
print(a.split())
Вывод**: ['itcoder', 'python'].**

Q69. Как импортировать модули в **python?**

Ответ:

Модули можно импортировать с помощью ключевого слова **import.** Вы можете импортировать модули тремя способами**.**
Пример**:**
import array # импорт с использованием оригинального имени модуля
import array as arr #импортирование с использованием псевдонима
from array import * #импортирует все**,** что присутствует в модуле **array**

Q70. В чем разница между глубоким и неглубоким копированием**?**

Ответ:

Неглубокое копирование используется при создании нового типа экземпляра и сохраняет значения**,** которые копируются в новый экземпляр**.** Неглубокое копирование используется для копирования указателей ссылок так же**,** как и для копирования значений**.** Эти ссылки указывают на исходные объекты**,** и изменения**,** внесенные в любой член класса**,** также повлияют на его исходную копию**.** Мелкое копирование позволяет ускорить выполнение программы и зависит от размера используемых данных**.**

Глубокое копирование используется для хранения значений**,** которые уже скопированы**.** При глубоком копировании не копируются указатели ссылок на объекты**.** Она делает ссылку на объект**,** а новый объект**,** на который указывает другой объект**,** получает сохранение**.** Изменения**,** внесенные в исходную копию**,** не повлияют ни на одну другую копию**,** использующую этот объект**.** Глубокое копирование делает выполнение программы медленнее из**-**за создания определенных копий для каждого вызываемого объекта**.**

Далее в этой части книги "" давайте рассмотрим объектно-ориентированные концепции в **Python.**

Q70. Как создаются классы в **Python?**

Ответ:

Класс в **Python** создается с помощью ключевого слова **class.**

Пример**:**

```
class Employee:
        def __init__(self, name):
            self.name = name
E1=Employee("Makers")
print(E1.name)
```

Выходные данные**: Makers**

Q71. Что такое **" monkey patching"** в **Python?**

Ответ:

В **Python** термин "обезьяний патч" относится только к динамическим изменениям класса или модуля во время выполнения**.**

Рассмотрим следующий пример**:**

```
# file.py
class MyClass:
    def f(self):
        print("f()")
```

Затем мы можем запустить тестирование с помощью обезьяньего патча следующим образом**:**

```
import ex
def monkey_f(self):
    print("monkey_f()")

ex.MyClass.f = monkey_f
obj = ex.MyClass()
obj.f()
```

Вывод будет выглядеть следующим образом**:**

```
monkey_f()
```

Как мы видим**,** мы внесли некоторые изменения в поведение функции **f()** в **MyClass,** используя определенную нами функцию **monkey_f()** вне модуля **file.**

Q72. Поддерживает ли **python** множественное наследование**?**

Ответ:

Множественное наследование означает**,** что класс может быть получен от более чем одного родительского класса**. Python** поддерживает множественное наследование**,** в отличие от **Java.**

Q73. Что такое полиморфизм в **Python?**

Ответ:

Полиморфизм означает способность принимать несколько форм**.** Например**,** если в родительском классе есть метод с именем **ABC,** то в дочернем классе тоже может быть метод с тем же именем **ABC** со своими параметрами и переменными**. Python** позволяет использовать полиморфизм**.**

Q74. Дайте определение инкапсуляции в **Python?**

Ответ:

Инкапсуляция означает связывание кода и данных вместе**.** Класс в **Python** является примером инкапсуляции**.**

Q75. Как сделать абстракцию данных в **Python?**

Ответ:

Абстракция данных **-** это предоставление только необходимых деталей и сокрытие реализации от посторонних глаз**.** Этого можно достичь в **Python** с помощью интерфейсов и абстрактных классов**.**

Q76.Использует ли **Python** спецификаторы доступа**?**

Ответ:

Python не лишает доступа к переменной экземпляра или функции**.** В **Python** заложена концепция префиксации имени переменной**,** функции или метода одинарным или двойным подчеркиванием для имитации поведения спецификаторов доступа **protected** и **private.**

Q77. Как создать пустой класс в **Python?**

Ответ:

Пустой класс **-** это класс**,** который не имеет никакого кода**,** определенного в его блоке**.** Он может быть создан с помощью ключевого слова **pass.** Однако объекты этого класса можно создавать вне самого класса**.** В **PYTHON** команда **PASS** ничего не делает**,** когда выполняется**.** Это нулевой оператор**.**

Например**:**

```
class Myclass:
        pass
obj=Myclass()
obj.name="KG"
print(" Name = "obj.name")
```

Вывод**:**

Name = KG

Q78. Что делает функция **object()?**

Ответ:

Он возвращает не имеющий свойств объект**,** который является базой для всех классов**.** Кроме того**,** он не принимает никаких параметров**.**

Далее, давайте посмотрим на некоторые базовые программы **Python** в этих вопросах для собеседования по **Python.**

Q79. Напишите программу на языке **Python** для выполнения алгоритма сортировки пузырьком**.**

Ответ:

```python
list1=[32,5,3,6,7,54,87]

def bubbleSort( theSeq ):
    n = len( theSeq )
    for i in range( n - 1 ) :
        flag = 0
        for j in range(n - 1) :
            if theSeq[j] > theSeq[j + 1] :
                tmp = theSeq[j]
                theSeq[j] = theSeq[j + 1]
                theSeq[j + 1] = tmp
                flag = 1
        if flag == 0:
            break
    return theSeq

print (bubbleSort(list1))
```

Вывод**:**

[3, 5, 6, 7, 32, 54, 87]

Q80. Напишите программу на языке **Python** для получения треугольника **Star.**

Ответ:

```python
def pyfunc(r):
    for x in range(r):
        print(' '*(r-x-1)+'*'*(2*x+1))
pyfunc(9)
```

```
PROBLEMS    OUTPUT    TERMINAL
TERMINAL
(venv) →  exel python3 test.py
          *
         ***
        *****
       *******
      *********
     ***********
    *************
   ***************
  *****************
*******************
```

Q81. Напишите программу для получения ряда Фибоначчи на языке **Python.**

Ответ:

```
# Функция для n-го числа Фибоначчи
def Fibonacci(n):

    # Проверяет, если входные данные равны 0, то
    # выведет неверный ввод
    if n < 0:
        print("Неверный ввод")

    # Проверьте, если n равно 0.
    # тогда он вернет 0
    elif n == 0:
        return 0

    # Проверьте, если n равно 1,2
    # вернется 1
    elif n == 1 or n == 2:
        return 1
    else:
        return Fibonacci(n-1) + Fibonacci(n-2)
print(Fibonacci(9))
```

Вывод: **34**

Q82. Напишите программу на языке **Python** для проверки того, является ли число простым.

Ответ:

```python
# Для получения ввода от пользователя
num = int(input("Введите число: "))

# простые числа больше 1
if num > 1:
    # проверяем наличие факторов
    for i in range(2,num):
        if (num % i) == 0:
            print(num, "не является простым числом")
            break
    else:
        print(num, "является простым числом")

# если вводимое число меньше
# или равно 1, оно не является простым
else:
    print(num, "не является простым числом")
```

Q83. Напишите программу на языке **Python** для проверки того, является ли последовательность палиндромом.

Ответ:

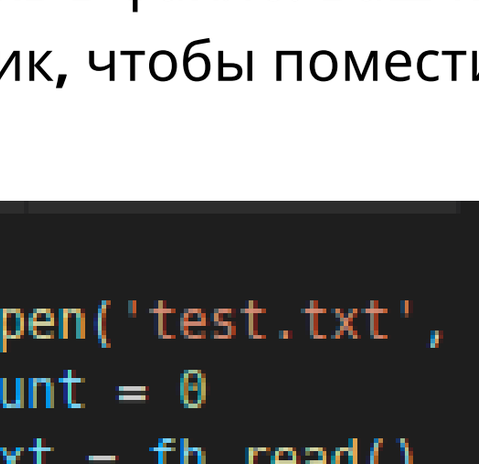
```
test.py > ...
1   a=input("enter sequence")
2   b=a[::-1]
3
4   if a==b:
5       print("palindrome")
6   else:
7       print("Not a Palindrome")
```

Q84. Напишите однострочный код, который подсчитывает количество заглавных букв в файле. Ваш код должен работать, даже если файл слишком велик, чтобы поместиться в памяти.

Ответ:

```
test.py > ...
1   with open('test.txt', 'r') as fh:
2       count = 0
3       text = fh.read()
4       for character in text:
5           if character.isupper():
6               count += 1
7           print(count)
```

При условии если содержимое файла test.txt :



```
test.txt
1   ПРИВЕТ

PROBLEMS   OUTPUT   TERMINAL

TERMINAL

(venv) →  exel python3 test.py
1
2
3
4
5
6
```

Теперь мы попытаемся преобразовать это в одну строку.

```
with open('test.txt', 'r') as fh:
    count=sum(1 for line in fh for character in line if character.isupper())
    print(count)
```

Q85. Напишите алгоритм сортировки для числового набора данных на языке **Python.**

Ответ:

Следующий код может быть использован для сортировки списка в **Python:**

```
list = ["1", "4", "0", "6", "9"]
list = [int(i) for i in list]
list.sort()
print (list)
```

Q83. Глядя на приведенный ниже код**,** запишите конечные значения **A0, A1, ...An.**

```
A0 = dict(zip(('a','b','c','d','e'),(1,2,3,4,5)))
A1 = range(10)A2 = sorted([i for i in A1 if i in A0])
A3 = sorted([A0[s] for s in A0])
A4 = [i for i in A1 if i in A3]
A5 = {i:i*i for i in A1}
A6 = [[i,i*i] for i in A1]
print(A0,A1,A2,A3,A4,A5,A6)
```

Ответ:

В качестве конечных выходов **A0, A1, ... A6** будут использоваться следующие значения**:**

```
A0 = {'a': 1, 'c': 3, 'b': 2, 'e': 5, 'd': 4} # порядок может варьироваться
A1 = range(0, 10)
A2 = []
A3 = [1, 2, 3, 4, 5]
A4 = [1, 2, 3, 4, 5]
A5 = {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}
A6 = [[0, 0], [1, 1], [2, 4], [3, 9], [4, 16], [5, 25], [6, 36], [7, 49], [8, 64], [9, 81]]
```

Q86. Объясните**,** что такое **Flask** и в чем его преимущества**?**

Ответ:

Flask - это веб**-**микрофреймворк для **Python,** основанный на лицензии **BSD "Werkzeug, Jinja2** и добрые намерения**". Werkzeug** и **Jinja2** являются двумя его зависимостями**.** Это означает**,** что у него практически не будет зависимостей от внешних библиотек**.** Это делает фреймворк легким**,** в нем мало зависимостей для обновления и меньше ошибок безопасности**.**

Сессия в основном позволяет запоминать информацию от одного запроса к другому**.** В **flask** сессия использует подписанный файл **cookie,** поэтому пользователь может просматривать содержимое сессии и изменять его**.** Пользователь может изменять сессию**,** если только у него есть секретный ключ **Flask.secret_key.**

Q87. Отличие **flask** от **django**

Ответ:

Flask намного проще по сравнению с **Django**, но **Flask** не делает многого за вас, то есть вам придется уточнять детали, в то время как **Django** делает многое за вас, и вам не придется прилагать много усилий. **Django** состоит из заранее написанного кода, который пользователю придется анализировать, в то время как **Flask** дает пользователям возможность создавать свой собственный код, что упрощает понимание кода.
Технически оба варианта одинаково хороши, и оба содержат свои плюсы и минусы.

Q88. Укажите различия между **Django**, **Pyramid** и **Flask.**

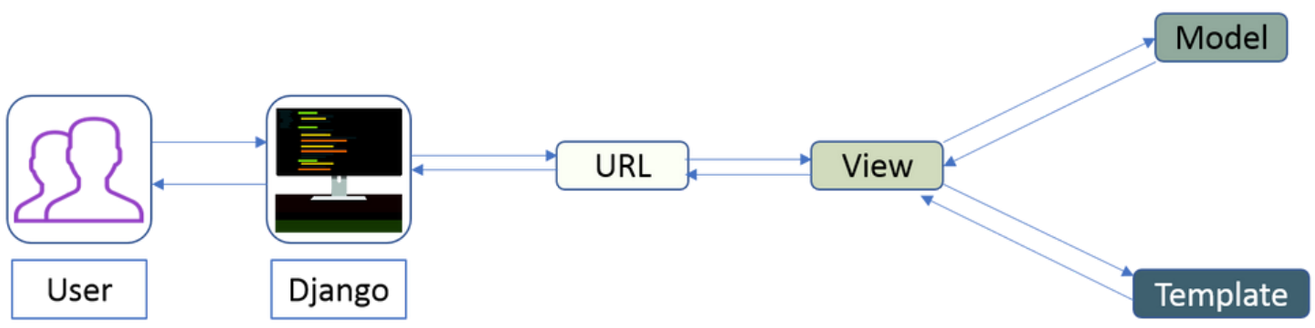
Ответ:

Flask - это "микрофреймворк", созданный в первую очередь для небольших приложений с более простыми требованиями. Во **Flask** вам придется использовать внешние библиотеки. **Flask** готов к использованию.
Pyramid создан для более крупных приложений. Она обеспечивает гибкость и позволяет разработчику использовать правильные инструменты для своего проекта. Разработчик может выбрать базу данных, структуру **URL**, стиль шаблонов и многое другое. **Pyramid** обладает широкими возможностями конфигурирования.
Django также можно использовать для больших приложений, как и **Pyramid.** Он включает в себя **ORM.**

Q89. Расскажите об архитектуре **Django.**

Ответ:

Django MVT Шаблон:



Разработчик предоставляет модель, представление и шаблон, затем просто сопоставляет их с **URL**, и **Django** делает волшебство, чтобы предоставить их пользователю.

Q90. Объясните, как вы можете настроить базу данных в **Django.**

Ответ:

Вы можете использовать команду редактировать **mysite/setting.py**, это обычный модуль **python** с уровнем модуля, представляющий настройки **Django.**

Django использует **SQLite** по умолчанию; это просто для пользователей **Django**, так как не требует установки каких-либо других типов. В случае, если ваш выбор базы данных отличается, вы должны изменить следующие ключи в пункте **DATABASE 'default',** чтобы они соответствовали вашим настройкам подключения к базе данных.

Engines: вы можете изменить базу данных, используя **'django.db.backends.sqlite3', 'django.db.backends.mysql', 'django.db.backends.postgresql_psycopg2', 'django.db.backends.oracle'** и т.д.

Name: Имя вашей базы данных. В случае, если вы используете **SQLite** в качестве базы данных, в этом случае база данных будет файлом на вашем компьютере, **Name** должно быть полным абсолютным путем, включая имя файла.

Если вы не выбираете **SQLite** в качестве базы данных, то необходимо добавить такие параметры, как Пароль, Хост, Пользователь и т.д.

Django использует **SQLite** в качестве базы данных по умолчанию, она хранит данные в виде одного файла в файловой системе. Если у вас есть сервер баз данных **- PostgreSQL, MySQL, Oracle, MSSQL -** и вы хотите использовать его, а не **SQLite**, то используйте инструменты администрирования вашей базы данных, чтобы создать новую базу данных для вашего проекта **Django.** В любом случае, с вашей **(**пустой**)** базой данных на месте, все, что остается **-** это указать **Django**, как ее использовать. Здесь на помощь приходит файл **settings.py** вашего проекта.

Мы добавим следующие строки кода в файл **settings.py:**

```
DATABASES = {
    'default': {
        'ENGINE' : 'django.db.backends.sqlite3',
        'NAME' : os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
```

Q91. Приведите пример, как можно написать **VIEW** в **Django?**

Ответ:

Вот как мы можем использовать запись представления в **Django:**

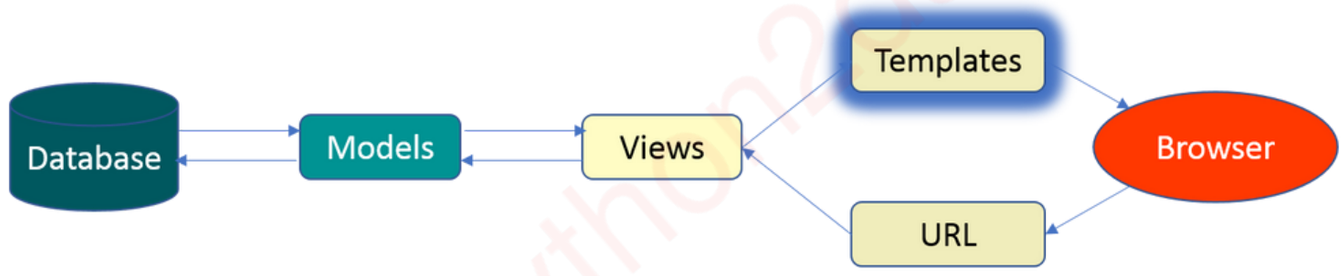
```
from django.http import HttpResponse
import datetime

def Current_datetime(request):
    now = datetime.datetime.now()
    html = "It is now %s/body/html" % now
    return HttpResponse(html)
```

Возвращает текущую дату и время в виде **HTML-**документа**.**

Q92. Укажите, из чего состоят шаблоны **Django.**

Ответ:

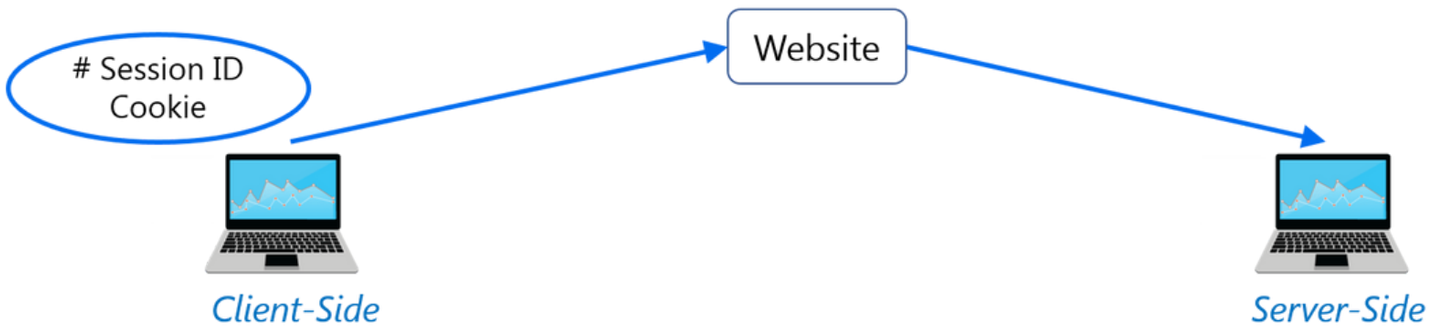
Шаблон **-** это простой текстовый файл**.** Он может создавать любой текстовый формат, например, **XML, CSV, HTML** и т.д**.** Шаблон содержит переменные, которые заменяются значениями при обработке шаблона, и теги **(% tag %),** которые управляют логикой шаблона**.**



Q91. Объясните использование сессии в фреймворке **Django?**

Ответ:

Django предоставляет сессию, которая позволяет хранить и извлекать данные на основе каждого посетителя сайта**. Django** позволяет абстрагироваться от процесса отправки и получения **cookies,** размещая **cookie** с идентификатором сессии на стороне клиента и сохраняя все связанные с ним данные на стороне сервера**.**



Таким образом, сами данные не хранятся на стороне клиента**.** Это хорошо с точки зрения безопасности**.**

Q93. Типы наследования в моделях **Django**

Ответ:

В **Django** существует три возможных типа наследования:

- Абстрактные базовые классы: Этот тип используется, когда вы хотите, чтобы только родительский класс хранил информацию, которую вы не хотите вводить для каждой дочерней модели.
- Мультитабличное наследование: Этот тип используется, если вы создаете подкласс существующей модели и хотите, чтобы каждая модель имела свою собственную таблицу базы данных.
- Прокси-модели: Вы можете использовать эту модель, если вы хотите изменить только поведение модели на уровне **Python**, не изменяя поля модели.

Q94. Как сохранить изображение локально с помощью **Python**, **URL-**адрес которого я уже знаю**?**

Ответ:

Мы будем использовать следующий код для локального сохранения изображения с **URL-**адреса

```
import urllib.request
urllib.request.urlretrieve("URL", "local-filename.jpg")
```

Q95. Как получить возраст кэша **Google** для любого **URL-**адреса или веб-страницы**?**

Ответ:

Используйте следующий формат **URL:**
http://webcache.googleusercontent.com/search?q=cache:URLGOESHERE

Обязательно замените **"URLGOESHERE"** на правильный веб-адрес страницы или сайта, кэш которого вы хотите получить и посмотреть время. Например, чтобы проверить возраст **Google Webcache** для сайта **edureka.co**, вы используете следующий **URL:**

http://webcache.googleusercontent.com/search?q=cache:edureka.co

Q96. Что такое функция **map** в **Python?**

Ответ:

Функция **map** выполняет функцию, заданную в качестве первого аргумента, над всеми элементами итерируемой группы элементов, заданной в качестве второго аргумента. Если функция принимает более **1** аргумента, то задается множество итераций.

Q97. Является ли **python numpy.array** лучше, чем списки?

Ответ:

Мы используем массив **python numpy** вместо списка по следующим трем причинам:

1. Меньше памяти
2. Быстро
3. Удобно

Q98. Как получить индексы **N** максимальных значений в массиве **NumPy?**

Ответ:

Мы можем получить индексы **N** максимальных значений в массиве **NumPy,** используя приведенный ниже код:

```
import numpy as np
arr = np.array([1, 3, 2, 4, 5])
print(arr.argsort()[-3:][::-1])
```

вывод:

[4 3 1]

Q99. Как вычислить процентную долю с помощью **Python/ NumPy?**

Ответ:

Мы можем рассчитать процентные доли с помощью следующего кода:

```
import numpy as np
a = np.array([1,2,3,4,5])
p = np.percentile(a, 50) #Returns 50th percentile, e.g. median
print(p)
```

вывод: **3**

Q100. В чем разница между **NumPy** и **SciPy?**

Numpy	**SciPy**
Относится к Числовому **python.**	Относится к **Scientific python.**
В нем меньше новых функций для научных вычислений**.**	Большинство новых функций научных вычислений относится к **SciPy.**
Содержит меньше функций линейной алгебры**.**	В нем есть более полнофункциональные версии модулей линейной алгебры**,** а также многие другие численные алгоритмы**.**
NumPy имеет более высокую скорость обработки**.**	**SciPy**, с другой стороны**,** имеет более низкую скорость вычислений**.**

Q101. Как с помощью **NumPy/SciPy** создавать **3D-**графики**/**визуализации**?**

Ответ:

Как и **2D** графики**, 3D** графика выходит за рамки **NumPy** и **SciPy,** но**,** как и в случае с **2D,** существуют пакеты**,** которые интегрируются с **NumPy. Matplotlib** обеспечивает базовое **3D** графику в подпакете **mplot3d,** а **Mayavi** предоставляет широкий спектр высококачественных функций **3D** визуализации**,** используя мощный движок **VTK.**

Q102. Какие из следующих утверждений создают словарь**?**
(Возможны несколько правильных ответов**)**

a) d = {}
b) d = {"john":40, "peter":45}
c) d = {40: "john", 45: "peter"}
d) d = (40: "john", 45: "50")

Ответ**: b, c** и **d.**

Словари создаются путем указания ключей и значений**.**

Q103. Что из этого является делением на половину**?**
a) /
b) //
c) %
d) Ни одно из перечисленных

Ответ**:** б**) //**
Когда оба операнда являются целыми числами**, python** вырезает дробную часть и выдает округленное значение**,** для получения точного ответа используйте деление на половину**.** Например**, 5/2 = 2,5,** но оба операнда целые**,** поэтому ответ этого выражения в **python** равен **2.**
Чтобы получить **2,5** в качестве ответа**,** используйте деление на пополам с помощью **//.** Итак**, 5//2 = 2,5**

Q104. Какова максимально возможная длина идентификатора**?**
a) 31 символ
b) 63 символа
c) 79 символов
г**)** Ничего из вышеперечисленного
Ответ**: d)** Ничего из вышеперечисленного

Идентификаторы могут быть любой длины**.**

Q105. Почему не рекомендуется использовать имена локальных переменных**,** начинающиеся с подчеркивания**?**
a) они используются для обозначения приватных переменных класса
b) они запутывают интерпретатор
c) они используются для обозначения глобальных переменных
d) они замедляют выполнение
Ответ**: a)** они используются для обозначения приватной переменной класса

Поскольку в **Python** нет понятия приватных переменных**,** ведущие символы подчеркивания используются для обозначения переменных**,** к которым нельзя обращаться извне класса**.**

Q106. Какое из следующих утверждений является неверным**?**

a) abc = 1 000 000

b) a b c = 1000 2000 3000

c) a,b,c = 1000, 2000, 3000

d) a_b_c = 1 000 000

Ответ**:** б**) a b c = 1000 2000 3000**

Пробелы не допускаются в именах переменных**.**

Q107. Что является результатом следующих действий**?**

```
try:
    if '1' != 1:
        raise "someError"
    else:
        print("someError has not occured")
except "someError":
    print ("someError has occured")
```

a) someError has occured

b) someError has not occured

c) invalid code

d) none of the above

Ответ**: c) invalid code**

Новый класс исключения должен наследоваться от **BaseException.** Здесь такого наследования нет**.**

Q108. Предположим**,** что **list1** имеет вид **[2, 33, 222, 14, 25],** чему равен **list1[-1]?**

a) Error

b) Нет

c) 25

d) 2

Ответ**: c) 25**

Индекс **-1** соответствует последнему индексу в списке**.**

Q109. Чтобы открыть файл **c:scores.txt** для записи**,** мы используем

a) outfile = open("c:scores.txt", "r")

b) outfile = open("c:scores.txt", "w")

c) outfile = open(file = "c:scores.txt", "r")

d) outfile = open(file = "c:scores.txt", "o")

Ответ:

b) Расположение содержит двойную косую черту **( )**, а **w** используется для указания того, что файл записывается в файл**.**

Q110. Каков выход следующих действий**?**

```
f = None

for i in range (5):
    with open("data.txt", "w") as f:
        if (i > 2):
            break

print (f.closed)
```

a) True

b) False

c) None

d) Error

Вывод**: a) True**

Оператор **WITH** при использовании с **open file** гарантирует**,** что объект **file** будет закрыт при выходе из блока **with.**

Q111. Когда будет выполняться часть **else** в **try-except-else?**

a) всегда

b) при возникновении исключения

c) когда исключение не возникает

d) когда исключение возникает в блоке **except**

Ответ**:** c**)** когда исключение не происходит

Часть **else** выполняется**,** когда исключение не происходит**.**

ООП-Объектно ориентированное программирование

ITcoder

Q1. В чем разница между **OOP** и **SOP?**

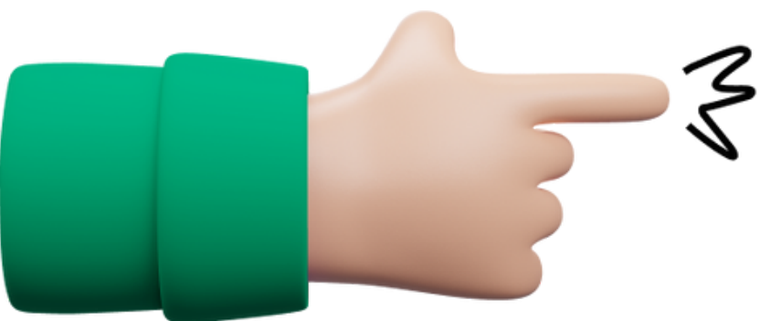
Ответ:

OOP-Объектно ориентированное программирование	**SOP-Структурное программирование**
Объектно-ориентированное программирование - это тип программирования, который основан на объектах, а не только на функциях и процедурном программировании.	Обеспечивает логическую структуру программы, где программы разделены на функции
Подход "снизу вверх"	Подход "сверху вниз
Обеспечивает сокрытие данных	Не обеспечивает сокрытие данных
Может решать задачи любой сложности	Может решать задачи средней сложности
Код можно использовать повторно, тем самым уменьшая избыточность	Не поддерживает возможность повторного использования кода

Q2. Что такое объектно-ориентированное программирование**?**

Ответ:

Объектно-ориентированное программирование (ООП) - это тип программирования, основанный на объектах, а не только на функциях и процедурном программировании. Отдельные объекты группируются в классы. ООП внедряет в программирование такие реальные сущности, как наследование, полиморфизм, скрытие и т.д. Оно также позволяет связывать данные и код вместе.



Q3. Зачем использовать ООП**?**

Ответ:

- ООП позволяет внести ясность в программирование**,** тем самым упрощая решение сложных проблем**.**
- Код можно использовать повторно через наследование**,** тем самым уменьшая избыточность
- Данные и код связаны вместе посредством инкапсуляции
- ООП позволяет скрывать данные**,** поэтому приватные данные остаются конфиденциальными
- Проблемы могут быть разделены на различные части**,** что упрощает их решение
- Концепция полиморфизма придает гибкость программе**,** позволяя сущностям иметь несколько форм

Q4. Каковы основные особенности ООП**?**

Ответ:

- Наследование
- Инкапсуляция
- Полиморфизм
- Абстракция

5. Что такое объект**?**

Ответ

Объект **-** это объект реального мира**,** который является базовой единицей ООП**,** например**,** стул**,** кошка**,** собака и т**.**д**.** Различные объекты имеют различные состояния или атрибуты**,** а также поведение**.**

6. Что такое класс**?**

Ответ

Класс **-** это прототип**,** состоящий из объектов в разных состояниях и с разным поведением**.** Он имеет ряд методов**,** которые являются общими для объектов**,** входящих в этот класс**.**

Q7 Разница между **__init__ ()** и **__new__ ()?**

Ответ:

new Вызывается при создании объекта, он возвращает экземпляр текущего объекта

init Вызывается после создания объекта, чтобы инициализировать некоторые экземпляры текущего объекта, нет возвращаемого значения

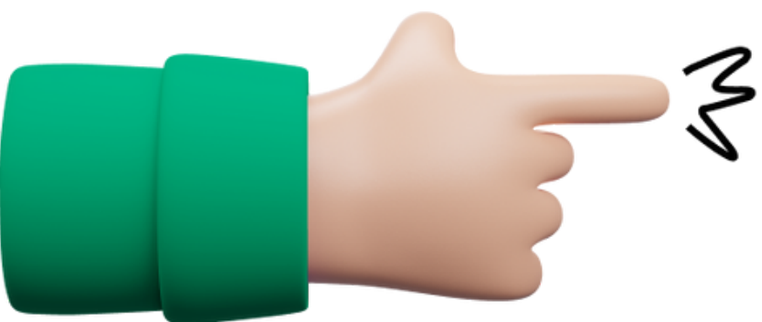
Сначала посмотрите на кусок кода:

```python
class Test(object):
    def __init__(self):

        print("__init__")
    def __new__(cls):

        print('__new__')
        return object.__new__(cls)

t = Test()
```

```
__new__
__init__

```

По результатам выполнения приведенного выше кода мы можем обнаружить, что программа сначала выполнила **__new__**, а затем выполнила **__init__**, что показывает, что в классе, если **__new__** и **__init__** существуют одновременно, сначала будет вызываться **__new__**.

- Метод **__new__** возвращает созданный объект, а **__init__ -** нет. **__init__** не имеет возвращаемого значения.
- У **__new__** должен быть хотя бы один параметр **cls,** представляющий класс **(**объект класса**),** для которого создается экземпляр.Этот параметр автоматически предоставляется интерпретатором **Python** при его создании.
- **__new__** должен иметь возвращаемое значение и возвращать инстанцированный экземпляр. Это реализовано самостоятельно. Обратите особое внимание на **__new__.** Вы можете вернуть экземпляр из родительского класса **__new__** или непосредственно из **__new__** объекта.
- **__init__** имеет параметр **self,** который является экземпляром, возвращаемым **__new__. __init__** может выполнить некоторые другие действия по инициализации на основе **__new__. __init__** не требует возвращаемого значения
- Мы можем сравнить аналогию с производителем: метод **__new__ -** это ссылка на покупку сырья на ранней стадии, а метод **__init__ -** для обработки и инициализации товарной ссылки на основе сырья.



Q8.Что такое **GIL?**

Ответ:

Python Global Interpreter Lock (GIL) — это своеобразная блокировка**,** позволяющая только одному потоку управлять интерпретатором **Python.** Это означает**,** что в любой момент времени будет выполняться только один конкретный поток**.**

Работа **GIL** может казаться несущественной для разработчиков**,** создающих однопоточные программы**.** Но во многопоточных программах отсутствие **GIL** может негативно сказываться на производительности процессоро**-**зависимых программ**.**

Поскольку **GIL** позволяет работать только одному потоку даже в многопоточном приложении**,** он заработал репутацию **«**печально известной**»** функции**.**

подробнее**: https://tproger.ru/translations/global-interpreter-lock-guide/**

Q9. В чем разница между классом и структурой**?**

Ответ:

Класс**:** Определяемая пользователем схема**,** на основе которой создаются объекты**.** Он состоит из методов или набора инструкций**,** которые должны быть выполнены над объектами**.**

Структура**:** Структура **-** это определенная пользователем коллекция переменных**,** которые имеют различные типы данных**.**

Q10. Можно ли вызвать метод базового класса без создания экземпляра**?**

Ответ:

Да**,** вы можете вызывать метод базового класса без создания его экземпляра**,** если**:**

Это статический метод
базовый класс наследуется каким**-**либо другим подклассом

Q11. В чем разница между классом и объектом**?**

Ответ:

Объект	**Класс**
Объект реального мира**,** являющийся экземпляром класса	Класс **-** это шаблон или схема**,** в рамках которой могут быть созданы объекты**.**
Объект действует как переменная класса	Связывает методы и данные в единое целое
Объект **-** это физическая сущность**.**	Класс **-** это логическая сущность**.**
Объекты занимают место в памяти при создании**.**	Класс не занимает место в памяти при создании**.**
Объекты могут быть объявлены по мере необходимости	Классы объявляются только один раз

Q12. Что такое наследование**?**

Ответ:

Наследование **-** это свойство ООП**,** которое позволяет классам наследовать общие свойства от других классов**.** Например**,** если есть класс **'vehicle',** то другие классы**,** такие как **'car', 'bike'** и т.д**.,** могут наследовать общие свойства от класса **vehicle.** Это свойство помогает избавиться от избыточного кода**,** тем самым уменьшая общий размер кода**.**

Q13. Каковы различные типы наследования**?**

Ответ:

- Одиночное наследование
- Множественное наследование
- Многоуровневое наследование
- Иерархическое наследование
- Гибридное наследование

Q14. В чем разница между множественным и многоуровневым наследованием**?**

Ответ:

Множественное наследование	**Многоуровневое наследование**
Множественное наследование возникает**,** когда класс наследует более одного базового класса	Многоуровневое наследование означает**,** что класс наследует от другого класса**,** который сам является подклассом другого базового класса**.**
Пример**:** Класс**,** определяющий ребенка**,** наследуется от двух базовых классов **Mother** и **Father**	Пример**:** Класс**,** описывающий спортивный автомобиль**,** наследуется от базового класса **Car,** который в свою очередь наследуется от другого класса **Vehicle.**

Q15. Что такое гибридное наследование**?**

Ответ:

Гибридное наследование **-** это сочетание множественного и многоуровневого наследования**.**

Q16. Что такое иерархическое наследование**?**

Ответ:

Иерархическое наследование относится к наследованию**,** при котором один базовый класс имеет более одного подкласса**.** Например**,** класс транспортного средства может иметь в качестве подклассов "автомобиль"**,** "велосипед" и т.д**.**

Q17. Каковы ограничения наследования**?**

Ответ:

- Увеличивает время и усилия**,** необходимые для выполнения программы**,** поскольку требуется переходить от одного класса к другому**.**
- Родительский класс и дочерний класс оказываются тесно связанными друг с другом
- Любые изменения в программе потребуют изменений как в родительском**,** так и в дочернем классе**.**
- Требуется тщательная реализация**,** иначе это приведет к неправильным результатам

Q18. Что такое суперкласс**(superclass)?**

Ответ:

Суперкласс или базовый класс **-** это класс**,** который выступает в роли родителя для какого**-**либо другого класса или классов**.** Например**,** класс **Vehicle** является суперклассом класса **Car.**

Q19. Что такое подкласс**?**

Ответ:

Класс**,** который наследуется от другого класса**,** называется подклассом**.** Например**,** класс **Car** является подклассом или производным от класса **Vehicle.**

Q20. Что такое полиморфизм**?**

Ответ:

Полиморфизм означает способность существовать в нескольких формах**.** Одному и тому же интерфейсу может быть дано несколько определений**.** Например**,** если у вас есть класс **Vehicle,** он может иметь метод **speed,** но вы не можете его определить**,** потому что у разных автомобилей разная скорость**.** Этот метод будет определен в подклассах с разными определениями для разных автомобилей**.**

Q21. Что такое статический полиморфизм**?**

Ответ:

Статический полиморфизм **(**статическое связывание**) -** это вид полиморфизма**,** который возникает во время компиляции**.** Примером полиморфизма во время компиляции является избыточность методов**.**

Q22. Что такое динамический полиморфизм**?**

Ответ:

Полиморфизм во время выполнения или динамический полиморфизм **(**динамическое связывание**) -** это тип полиморфизма**,** который определяется во время выполнения**.** Примером полиморфизма во время выполнения является переопределение методов**.**

Q23. Что такое избыточность методов**?**

Ответ:

Избыточность методов **-** это свойство ООП**,** которое позволяет давать одинаковое имя нескольким методам в классе**,** если передаваемые аргументы различны**.**

Q24. Что такое переопределение методов**?**

Ответ:

Переопределение методов **-** это свойство ООП**,** с помощью которого дочерний класс или подкласс может переопределять методы**,** присутствующие в базовом или родительском классе**.** При этом переопределяемый метод имеет то же имя**,** сигнатуру**,** означающую передаваемые аргументы и возвращаемый тип**.**

Q25. Что такое избыточность операторов**?**

Ответ:

Избыточность операторов **-** это реализация операторов**,** использующих определенные пользователем типы**,** на основе переданных вместе с ними аргументов**.**

Q26. Проведите различие между перегрузкой и переопределением.

Ответ:

Перегрузка	**Переопределение**
Два или более метода с одинаковым именем, но разными параметрами или сигнатурой	Дочерний класс переопределяет методы, присутствующие в базовом классе, с одинаковыми параметрами/ сигнатурой
Решается во время компиляции	Решается во время выполнения

Q27. Что такое инкапсуляция?

Ответ:

Инкапсуляция - это связывание данных и кода, который работает с ними, в единое целое. Например, класс. Инкапсуляция также позволяет скрывать данные, поскольку данные, указанные в одном классе, скрыты от других классов.

Q28. Что такое "спецификаторы доступа"?

Ответ:

Спецификаторы доступа или модификаторы доступа - это ключевые слова, которые определяют доступность методов, классов и т.д. в ООП. Эти спецификаторы доступа позволяют реализовать инкапсуляцию. Наиболее распространенными спецификаторами доступа являются **public**, **private** и **protected**. Однако существует еще несколько, специфичных для языков программирования.

Q29. В чем разница между модификаторами доступа **public, private** и **protected?**

Ответ:

Name	**Доступность из собственного класса**	**Доступность из производного класса**	**Доступность со всего мира**
public	Да	Да	Да
private	Да	Нет	Нет
protected	Да	Да	Нет

Q30. Что такое абстракция **?**

Ответ:

Абстракция **-** это очень важная особенность ООП**,** которая позволяет отображать только важную информацию и скрывать детали реализации**.** Например**,** катаясь на велосипеде**,** вы знаете**,** что если поднять педаль газа**,** то скорость увеличится**,** но не знаете**,** как это происходит на самом деле**.** Это абстракция данных**,** поскольку детали реализации скрыты от велосипедиста**.**

Q31. Как достичь абстракции данных**?**

Ответ:

Абстракция данных может быть достигнута с помощью**:**

- Абстрактный класс
- Абстрактный метод

Q32. Что такое абстрактный класс**?**

Ответ:

Абстрактный класс **-** это класс**,** который состоит из абстрактных методов**.** Эти методы в основном объявлены**,** но не определены**.** Если эти методы будут использоваться в каком**-**то подклассе**,** то они должны быть определены исключительно в этом подклассе**.**

Q33. Можете ли вы создать экземпляр абстрактного класса**?**

Ответ:

Нет**.** Экземпляры абстрактного класса не могут быть созданы**,** поскольку он не имеет полной реализации**.** Однако экземпляры подкласса**,** наследующего абстрактный класс**,** могут быть созданы**.**

Q34. Что такое интерфейс**?**

Ответ:

Это концепция ООП**,** которая позволяет объявлять методы**,** не определяя их**.** Интерфейсы**,** в отличие от классов**,** не являются чертежами**,** поскольку не содержат подробных инструкций или действий**,** которые должны быть выполнены**.** Любой класс**,** реализующий интерфейс**,** определяет методы интерфейса**.**

Q35. Проведите различие между абстракцией и инкапсуляцией**.**

Ответ:

Абстракция	**Инкапсуляция**
Решает проблему на уровне проектирования	Решает проблему на уровне реализации
Позволяет показать важные аспекты**,** скрывая детали реализации	Связывает код и данные в единое целое и скрывает его от посторонних глаз

36. Что такое конструктор**?**

Ответ:

Конструктор **-** это особый тип метода**,** который имеет то же имя**,** что и класс**,** и используется для инициализации объектов этого класса**.**

37. Что такое деструктор**?**

Ответ:

Деструктор **-** это метод**,** который автоматически вызывается при уничтожении объекта**.** Деструктор также восстанавливает пространство памяти**,** которое было выделено уничтоженному объекту**,** закрывает файлы и соединения базы данных объекта и т**.**д**.**

38. Типы конструкторов

Ответ:

Типы конструкторов различаются в зависимости от языка**.** Тем не менее**,** все возможные конструкторы существуют**:**

- Конструктор по умолчанию
- Параметризованный конструктор
- Копирующий конструктор
- Статический конструктор
- Частный конструктор

39. Что такое конструктор копирования**?**

Ответ:

Конструктор копирования создает объекты путем копирования переменных из другого объекта того же класса**.** Основная цель конструктора копирования **-** создать новый объект из существующего**.**

40. Для чего используется **'finalize'?**

Ответ:

Finalize как метод объекта используется для освобождения неуправляемых ресурсов и очистки перед сборкой мусора **(GC).** Он выполняет задачи управления памятью**.**

41. Что такое сборка мусора **(GC)?**

Ответ:

GC - это реализация автоматического управления памятью**.** Сборщик мусора освобождает место**,** занимаемое объектами**,** которые больше не существуют**.**

42. Проведите различие между классом и методом**.**

Ответ:

Класс	**Метод**
Класс **-** это шаблон**,** который связывает код и данные в единое целое**.** Классы состоят из методов**,** переменных и т.д	Вызываемый набор инструкций**,** также называемый процедурой или функцией**,** которые должны быть выполнены над заданными данными**.**

Q43. Что такое конечная переменная**?**

Ответ:

Переменная**,** значение которой не изменяется**.** Она всегда ссылается на один и тот же объект по свойству нетрансверсивности**.**

Q44. Что такое исключение**?**

Ответ:

Исключение **-** это вид уведомления**,** которое прерывает нормальное выполнение программы**.** Исключения предоставляют образец ошибки и передают ошибку обработчику исключений для ее разрешения**.** Состояние программы сохраняется**,** как только возникает исключение**.**

Q45. Что такое обработка исключений**?**

Ответ:

Обработка исключений в объектно**-**ориентированном программировании **-** это очень важная концепция**,** которая используется для управления ошибками**.** Обработчик исключений позволяет бросать и ловить ошибки и реализует централизованный механизм для их разрешения**.**

Q46. Что такое блок **try/ catch?**

Ответ:

Блок **try/ catch** используется для обработки исключений**.** Блок **try** определяет набор утверждений**,** которые могут привести к ошибке**.** Блок **catch** в основном перехватывает исключение**.**

Q47. Что такое блок **finally?**

Ответ:

Блок **finally** состоит из кода**,** который используется для выполнения важного кода**,** такого как закрытие соединения и т**.** д**.** Этот блок выполняется**,** когда блок **try** завершается**.** Он также гарантирует**,** что блок **finally** будет выполнен даже в случае возникновения неожиданного исключения**.**

Q48. Каковы ограничения ООП**?**

Ответ:

Обычно не подходит для решения небольших задач

Требуется интенсивное тестирование

Требуется больше времени для решения проблемы

Требует надлежащего планирования

Программист должен думать о решении проблемы в терминах объектов

Q1. В чем разница между **Flask** и **Django?**

Ответ:

Коэффициент сравнения	**Django**	**Flask**
Тип проекта	Поддерживает крупные проекты	Создан для небольших проектов
Шаблоны, администрирование и **ORM**	Встроенный	Требуется установка
Легкость обучения	Требуется дополнительное обучение и практика	Легко научиться
Гибкость	Позволяет осуществлять полную веб-разработку без необходимости использования сторонних инструментов	Более гибкий, поскольку пользователь может выбрать любой сторонний инструмент в соответствии со своим выбором и требованиями
Визуальная отладка	Не поддерживает визуальную отладку	Поддерживает визуальную отладку
Тип фреймворка	Много библиотек, и инструментов	Простой и легкий
Инструмент **Bootstrap**	Встроенный	Недоступен

Django

ITcoder

Q1. Что такое **Django?**

Ответ:

Django - это платформа для разработки веб-сайтов, которая была разработана в быстро развивающейся редакции новостей. Это бесплатная платформа с открытым исходным кодом, которая была названа в честь Джанго Рейнхардта, джазового гитариста **1930-**х годов. **Django** поддерживается некоммерческой организацией под названием **Django Software Foundation.** Основная цель **Django -** обеспечить быструю и легкую веб-разработку.

Q2. Назовите некоторые компании, которые используют **Django?**

Ответ:

Некоторые из компаний, использующих **Django**, это **Instagram**, **DISCUS**, **Mozilla Firefox**, **YouTube**, **Pinterest**, **Reddit** и т.д.

Q3. Каковы особенности **Django?**

Ответ:

- **SEO-**оптимизированный
- Чрезвычайно быстрый
- Полностью нагруженный фреймворк, который поставляется вместе с аутентификацией, администрированием контента, **RSS-**каналами и т.д.
- Очень безопасен, что помогает разработчикам избежать распространенных ошибок безопасности, таких как подделка межсайтовых запросов **(csrf)**, **clickjacking**, межсайтовый скриптинг и т.д.
- Исключительная масштабируемость, что, в свою очередь, помогает удовлетворить самые высокие требования к трафику.
- Огромная универсальность, которая позволяет разрабатывать любые веб-сайты

Q4.Что вы знаете о **csrf_token?**

Ответ:

Токен **csrf_token** используется для защиты от межсайтовых подделок запросов**.** Этот вид атаки происходит**,** когда вредоносный веб**-**сайт состоит из ссылки**,** некоторого **JavaScript** или формы**,** целью которых является выполнение определенного действия на вашем сайте**,** используя учетные данные настоящего пользователя**.**

Q5. Как проверить версию **Django,** установленную в вашей системе**?**

Ответ:

Чтобы проверить версию **Django,** установленную в вашей системе**,** вы можете открыть командную строку и ввести следующую команду**:**

python -m django -version

Вы также можете попробовать импортировать **Django** и использовать метод **get_version()** следующим образом**:**

```
1. import django
2. print(django.get_version())
```

Q6. Каковы преимущества использования **Django?**

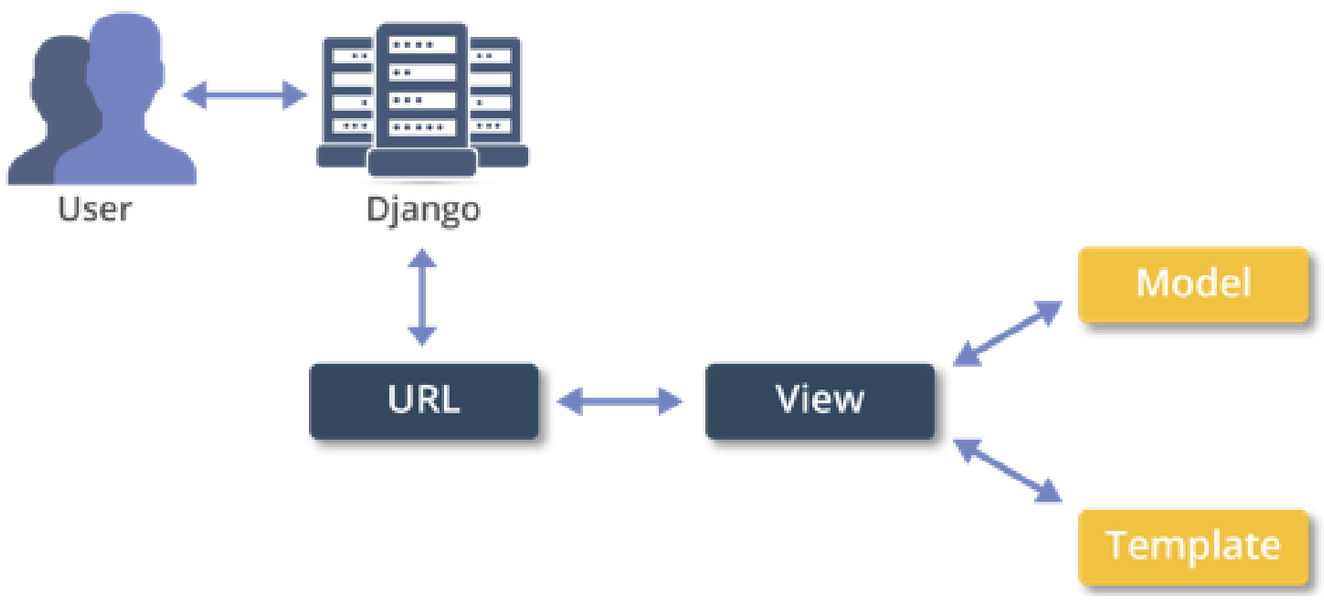
Ответ:

- Стек **Django** является свободно связанным с высокой сплоченностью**.**
- В приложениях **Django** используется очень мало кода
- Позволяет быстро разрабатывать веб**-**сайты
- Следуют принципу **DRY** или **"**Не повторяй себя**",** что означает**,** что одна концепция или часть данных должна жить только в одном месте
- Последовательность как на низких**,** так и на высоких уровнях
- Поведение не предполагается неявно**,** а указывается явно
- **SQL-**запросы не выполняются слишком много раз и оптимизируются внутри системы
- Можно легко перейти к необработанному **SQL,** когда это необходимо
- Гибкость при использовании **URL**

Q7. Объясните архитектуру **Django.**

Ответ:

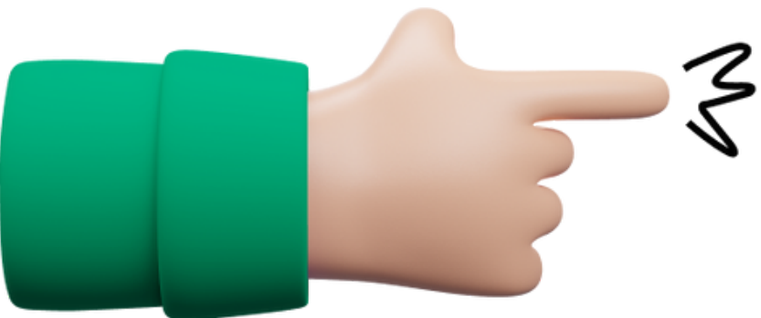
Django следует архитектуре **MVT** или **Model View Template,** которая основана на архитектуре **MVC** или **Model View Controller.** Основное различие между ними заключается в том**,** что **Django** сам заботится о части контроллера**.**



Согласно **Django**, "представление" в основном описывает данные**,** представленные пользователю**.** Он не имеет отношения к тому**,** как выглядят данные**,** а скорее к тому**,** что они собой представляют**.** Представления**,** по сути**,** являются функциями обратного вызова для указанных **URL,** и эти функции обратного вызова описывают**,** какие данные будут представлены**.**

С другой стороны**,** "шаблоны" имеют дело с представлением данных**,** тем самым отделяя содержание от его представления**.** В **Django** представления делегируют шаблонам полномочия по представлению данных**.**

"Контроллером" здесь является сам **Django,** который отправляет запрос на соответствующее представление в соответствии с указанным **URL.** Именно поэтому **Django** относят к архитектуре **MTV,** а не **MVC.**



Q8. Дайте краткую информацию о **'django-admin'.**

Ответ:

django-admin - это утилита командной строки **Django** для выполнения административных задач**.** С помощью **django-admin** вы можете выполнить ряд задач**,** некоторые из которых перечислены в следующей таблице**:**

Задачи	**Команды**
Для отображения информации об использовании и списка команд**,** предоставляемых каждым приложением	**django-admin help**
Чтобы отобразить список доступных команд	**django-admin help –command**
Чтобы отобразить описание данной команды и список ее доступных опций	**django-admin help <command>**
Определение версии **Django**	**django-admin version**
Создание новых миграций на основе изменений**,** внесенных в модели	**django-admin makemigrations**
Синхронизация состояния базы данных с текущим набором моделей и миграций	**django-admin migrate**
Запуск сервера разработки	**django-admin runserver**
Отправка тестового письма для подтверждения того**,** что отправка писем через **Django** работает	**django-admin sendtestemail**
Запуск интерактивного интерпретатора **Python**	**django-admin shell**
Чтобы показать все миграции в вашем проекте	**django-admin showmigrations**

Q9. Как подключить ваш проект **Django** к базе данных**?**

Ответ:

Django поставляется с базой данных по умолчанию**,** которая является **SQLite.** Чтобы подключить ваш проект к этой базе данных**,** используйте следующие команды**:**

- **python manage.py migrate (**команда **migrate** смотрит на настройки **INSTALLED_APPS** и создает таблицы базы данных соответствующим образом**)**
- **python manage.py makemigrations (**сообщает **Django,** что вы создали**/**изменили свои модели**)**
- **python manage.py sqlmigrate <**имя приложения**,** за которым следует сгенерированный **id> (sqlmigrate** берет имена миграций и возвращает их **SQL)**
-

Q10. Какие различные файлы создаются при создании проекта **Django?** Объясните кратко**.**

Ответ:

Когда вы создаете проект с помощью команды **startproject,** будут созданы следующие файлы**:**

Файлы	**Описание**
manage.py	Утилита командной строки**,** которая позволяет вам взаимодействовать с вашим проектом **Django**
__init__.py	Пустой файл**,** который сообщает **Python,** что текущий каталог должен рассматриваться как пакет **Python**
settings.py	Состоит из настроек для текущего проекта
urls.py	Содержит **URL-**адреса для текущего проекта
wsgi.py	Это точка входа для веб**-**серверов для обслуживания созданного вами проекта

Q11. Что такое **" Models"?**

Ответ:

Модели **-** это единый и окончательный источник информации о ваших данных**.** Она состоит из всех основных полей и моделей поведения данных**,** которые вы храните**.** Часто каждая модель сопоставляется с одной конкретной таблицей базы данных**.**

В **Django** модели служат в качестве слоя абстракции**,** который используется для структурирования и манипулирования вашими данными**.** Модели **Django** являются подклассом класса **django.db.models.Model,** а атрибуты в моделях представляют собой поля базы данных**.**

Q12. Что такое **" views"?**

Ответ:

Представления**(views)** в **Django** служат цели инкапсуляции**.** Они инкапсулируют логику**,** ответственную за обработку запроса пользователя и за возвращение ответа обратно пользователю**.** Представления в **Django** либо возвращают **HttpResponse,** либо вызывают исключение**,** например**, Http404. HttpResponse** содержит объекты**,** состоящие из содержимого**,** которое должно быть отображено пользователю**.** Представления также могут использоваться для выполнения таких задач**,** как чтение записей из базы данных**,** делегирование полномочий шаблонам**,** генерация **PDF-**файла и т.д**.**

Q13. Что такое **'templates'?**

Ответ:

templates(шаблоны**) Django** визуализирует информацию**,** которая должна быть представлена пользователю в удобном для дизайнера формате**.** Используя шаблоны**,** вы можете генерировать **HTML** динамически**. HTML** состоит как из статических**,** так и динамических частей контента**.** Вы можете иметь любое количество шаблонов в зависимости от требований вашего проекта**.** Можно обойтись и без них**.**

Django имеет собственную систему шаблонов**,** называемую языком шаблонов **Django (DTL).** Независимо от бэкенда**,** вы можете загружать и отображать шаблоны**,** используя стандартную админку **Django.**

Q14. В чем разница между проектом и приложением**?**

Ответ:

Приложение **-** это, по сути, веб-приложение, которое создается для того, чтобы что-то делать, например, база данных записей о сотрудниках. Проект, с другой стороны, представляет собой набор приложений какого-то конкретного сайта. Поэтому один проект может состоять из **'n'** количества приложений, а одно приложение может находиться в нескольких проектах.

Q15. Каковы различные виды наследования в **Django?**

Ответ:

В **Django** есть три возможных вида наследования:

Вид наследования	**Описание**
Абстрактные базовые классы	Используется, когда вы хотите использовать родительский класс для хранения информации, которую вы не хотите вводить для каждой дочерней модели. В данном случае родительский класс никогда не используется в одиночку.
Мультитабличное наследование	Используется, когда вам нужно подклассифицировать существующую модель и вы хотите, чтобы каждая модель имела свою таблицу базы данных. чтобы каждая модель имела свою собственную таблицу базы данных
Proxy модели	Используется, если вы хотите изменить только поведение модели на уровне **Python,** никак не изменяя поля **'models'.**

Q16. Что такое статические файлы**?**

Ответ:

Статические файлы в **Django -** это те файлы**,** которые служат целям дополнительных файлов**,** таких как **CSS,** изображения или файлы **JavaScript.** Этими файлами управляет **django.contrib.staticfiles.** Эти файлы создаются в директории приложения проекта путем создания поддиректории с именем **static.**

Q17. Что такое 'сигналы'**?**

Ответ:

Django состоит из диспетчера сигналов**,** который позволяет разделенным приложениям получать уведомления о действиях**,** происходящих в других частях фреймворка**. Django** предоставляет набор встроенных сигналов**,** которые в основном позволяют отправителям уведомлять набор получателей**,** когда выполняется какое**-**то действие**.** Некоторые из этих сигналов следующие**:**

Сигналы	**Описание**
django.db.models.signals.pre_save **django.db.models.signals.post_save**	Отправляется до или после вызова метода **save()** модели
django.db.models.signals.pre_deletedj **ango.db.models.signals.post_delete**	Отправляется до или после вызова метода **delete()** модели или метода **delete() queryset'**а
django.db.models.signals.m2m_changed	Отправляется**,** когда **Django** начинает или заканчивает **HTTP** запрос

Q18. Кратко объясните класс **Django Field.**

Ответ:

'Field' - это абстрактный класс**,** который фактически представляет столбец в таблице базы данных**.** Класс **Field,** в свою очередь**,** является подклассом **RegisterLookupMixin.** В **Django** эти поля используются для создания таблиц базы данных **(db_type()),** которые используются для отображения типов **Python** в базу данных с помощью **get_prep_value()** и наоборот с помощью метода **from_db_value().** Таким образом**,** поля являются фундаментальными элементами в различных **API Django,** таких как модели и наборы запросов**.**

Q19. Как создать проект **Django?**

Ответ:

Чтобы создать проект **Django,** зайдите в директорию**,** где вы хотите создать свой проект**,** и введите следующую команду**:**

django-admin startproject ITC

ПРИМЕЧАНИЕ**:** Здесь **ITC -** это имя проекта**.** Вы можете дать любое имя**,** какое пожелаете**.**

Q20. Что такое **mixin?**

Ответ:

Mixin - это тип множественного наследования**,** при котором вы можете объединять поведение и атрибуты более чем одного родительского класса**.** Миксины обеспечивают отличный способ повторного использования кода из нескольких классов**.** Например**,** общие представления на основе классов состоят из миксина под названием **TemplateResponseMixin,** целью которого является определение метода **render_to_response().** Когда он объединяется с классом**,** присутствующим в представлении**,** результатом будет класс **TemplateView.**

Недостатком использования этих миксинов является то**,** что становится трудно анализировать**,** что делает дочерний класс и какие методы следует переопределить**,** если его код слишком разбросан между несколькими классами**.**

Q21. Что такое **'sessions'?**

Ответ:

Сессии полностью поддерживаются в **Django.** Используя фреймворк сессий, вы можете легко хранить и извлекать произвольные данные, основанные на посетителях каждого сайта. Этот фреймворк в основном хранит данные на стороне сервера и заботится об отправке и получении **cookies.** Эти куки содержат идентификатор сессии, но не сами данные, если вы явно не используете бэкенд на основе куки.

Q22. Что такое контекст в **django** шаблонах**?**

Ответ:

Контекст в **Django -** это словарь отображения шаблонных переменных имен, присвоенных объектам **Python.** Это общепринятое название, но вы можете дать любое другое имя по своему выбору, если хотите это сделать.

Q23. Когда вы можете использовать итераторы в **Django ORM?**

Ответ:

Итераторы в **Python -** это, по сути, контейнеры, состоящие из счетного количества элементов. Любой объект, являющийся итератором, реализует два метода: **__init__()** и **__next__().** Когда вы используете итераторы в **Django**, лучше всего это делать, когда вам нужно обработать результаты, требующие большого объема памяти. Для этого вы можете использовать метод **iterator(),** который в основном оценивает **QuerySet** и возвращает соответствующий итератор над результатами.

Q24. Объясните стратегии кэширования в **Django?**

Ответ:

Кэширование в основном означает сохранение результатов какого-либо затратного вычисления, чтобы избежать повторного выполнения того же вычисления. **Django** предоставляет надежную систему кэширования, которая, в свою очередь, помогает вам сохранять динамические веб-страницы так, чтобы их не приходилось обрабатывать снова и снова для каждого запроса. Некоторые из стратегий кэширования **Django** перечислены в следующей таблице:

Стратегии	Описание
Memcached	Кэш-сервер на основе памяти, который является самым быстрым и эффективным.
Кэширование файловой системы	Значения кэша хранятся в виде отдельных файлов в последовательном порядке
Кэширование в локальной памяти	Это кэш по умолчанию, если вы не указали другой. Этот тип кэширования является перпроцессным и потокобезопасным.
Кэширование в базе данных	Данные кэша хранятся в базе данных и работают очень хорошо, если у вас быстрый и хорошо индексируемый сервер баз данных.

Q25. Объясните использование **Middlewares** в **Django.**

Ответ:

Middleware - это фреймворк, который представляет собой легкую и низкоуровневую систему плагинов для глобального изменения входных и выходных данных **Django.** По сути, это система крючков для обработки запросов/ответов в **Django.** Каждый компонент в **middleware** имеет определенную задачу. Например, **AuthenticationMiddleware** используется для привязки пользователей к запросам с помощью сессий. **Django** предоставляет множество других промежуточных программ, таких как **cache middleware** для включения кэша всего сайта, общая промежуточная программа, которая выполняет множество задач, таких как запрет доступа для агентов пользователя, перезапись **URL** и т.д., **GZip middleware**, которая используется для сжатия содержимого для браузеров и т.д.

Q26. Каково значение файла **manage.py** в **Django?**

Ответ:

Файл **manage.py** автоматически генерируется всякий раз, когда вы создаете проект. По сути, это утилита командной строки, которая помогает вам взаимодействовать с вашим проектом **Django** различными способами. Она делает то же самое, что и **django-admin**, но вместе с этим она также устанавливает переменную окружения **DJANGO_SETTINGS_MODULE**, чтобы указать на настройки вашего проекта. Обычно лучше использовать **manage.py**, а не **django-admin**, если вы работаете над одним проектом.

Q27. Объясните использование команды **'migrate'** в **Django?**

Ответ:

В **Django** миграции используются для распространения изменений, внесенных в модели. Команда **migrate** в основном используется для применения или неприменения изменений, сделанных миграциями, к моделям. Эта команда в основном синхронизирует текущий набор моделей и миграций с состоянием базы данных. Вы можете использовать эту команду с параметрами или без них. Если вы не укажете никаких параметров, во всех приложениях будут запущены все миграции.

Q28. Как просматривать и фильтровать элементы из базы данных**?**

Ответ:

Чтобы просмотреть все элементы из базы данных, вы можете воспользоваться функцией **'all()'** в интерактивной оболочке следующим образом:
ITC.objects.all(), где **ITC -** некоторый класс, который вы создали в своих моделях.
Чтобы отфильтровать некоторый элемент из вашей базы данных, вы можете использовать либо метод **get()**, либо метод **filter** следующим образом:
ITC.objects.filter(pk=1)
ITC.objects.get(id=1)

Q29. Объясните, как обрабатывается запрос в **Django?**

Ответ:

Если пользователь запрашивает страницу с какого-либо сайта на **Django**, система следует алгоритму, который определяет, какой **Python-**код должен быть выполнен. Вот шаги, которые составляют алгоритм:

Ugaru mebe!

ITcoder