1. В чем разница между списком и кортежем?

Мне задавали этот вопрос буквально на каждом собеседовании по Python/data science. Выучите ответ как свои пять пальцев:

1. Список можно изменить после создания.
2. Кортеж нельзя изменить после создания.
3. Список упорядочен. Он представляет собой упорядоченные последовательности объектов, как правило, одного и того же типа. Например, все имена пользователей упорядочены по дате создания: ["Seth", "Ema", "Eli"].
4. У кортежа есть структура. В каждом индексе могут сосуществовать различные типы данных. Например, такая запись базы данных в памяти: (2, "Ema", "2020–04–16") # id, name, created\_at.

2. Как выполняется интерполяция строк?

Без импорта класса Template есть три способа интерполяции строк:

name = 'Chris'  
  
# 1. f strings  
print(f'Hello {name}')  
  
# 2. % operator  
print('Hey %s %s' % (name, name))  
  
# 3. format  
print(  
 "My name is {}".format((name))  
)

3. В чем разница между is и ==?

Когда я был начинающим разработчиком, то не видел разницы… привет, баги. Так что для протокола: is проверяет идентичность, а == проверяет равенство.  
  
Рассмотрим пример. Создайте несколько списков и назначьте им имена. Обратите внимание, что ниже b указывает на тот же объект, что и a:

a = [1,2,3]  
b = a  
c = [1,2,3]

Проверьте равенство и обратите внимание, что все объекты равны:

print(a == b)  
print(a == c)  
#=> True  
#=> True

Но являются ли все они идентичными? Нет:

print(a **is** b)  
print(a **is** c)  
#=> True  
#=> False

Можем проверить это, распечатав их идентификаторы объектов:

print(id(a))  
print(id(b))  
print(id(c))  
#=> 4369567560  
#=> 4369567560  
#=> 4369567624

Идентификатор c отличается от идентификатора a и b.

4. Что такое декоратор?

Еще один вопрос, который мне задавали на каждом собеседовании. Тема заслуживает отдельной статьи, но для базовой подготовки достаточно просто написать собственный пример.  
  
Декоратор позволяет добавить новую функциональность к существующей функции. Это делается следующим образом. Функция передается декоратору, а он выполняет и существующий, и дополнительный код.  
  
Напишем декоратор, который записывает в журнал вызовы другой функции.  
  
**Напишите функцию декоратора**. В качестве аргумента он принимает функцию func. Декоратор определяет функцию log\_function\_called, которая вызывает func() и выполняет некоторый код print(f'{func} called.'). Затем возвращает определенную им функцию:

**def** logging(func):  
 def log\_function\_called():  
 print(f'{func} called.')  
 func()  
 return log\_function\_called

Напишем другие функции, к которым добавим декоратор (потом, не сейчас):

**def** my\_name():  
 print('chris')  
**def** friends\_name():  
 print('naruto')  
my\_name()  
friends\_name()  
#=> chris  
#=> naruto

Теперь добавим декоратор к ним обоим:

@logging  
**def** my\_name():  
 print('chris')  
  
@logging  
**def** friends\_name():  
 print('naruto')  
  
my\_name()  
friends\_name()  
#=> <function my\_name at 0x10fca5a60> called.  
#=> chris  
#=> <function friends\_name at 0x10fca5f28> called.  
#=> naruto

Теперь легко добавить ведение журнала в любую функцию, которую мы пишем. Достаточно написать перед ней @logging.

5. Объясните функцию range

Range генерирует список целых чисел. Ее можно использовать тремя способами.  
  
Функция принимает от одного до трех аргументов. Обратите внимание, что я завернул каждый пример в список, чтобы видеть генерируемые значения.  
  
**range(stop)** — генерирует целые числа от 0 до целого числа stop:

[i **for** i **in** range(10)]  
#=> [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

**range(start, stop)** — генерирует целые числа от start до stop:

[i **for** i **in** range(2,10)]  
#=> [2, 3, 4, 5, 6, 7, 8, 9]

**range(start, stop, step)** — генерирует целые числа от start до stop с интервалами step:

[i **for** i **in** range(2,10,2)]  
#=> [2, 4, 6, 8]

[Серж Боремчук](https://medium.com/@Searge) предложил более подходящий способ:

list(range(2,10,2))  
#=> [2, 4, 6, 8]

6. Определите класс car с двумя атрибутами: color и speed. Затем создайте экземпляр и верните speed

Вот как это сделать:

**class** Car :  
 **def** \_\_init\_\_(self, color, speed):  
 self.color = color  
 self.speed = speed  
car = Car('red','100mph')  
car.speed  
#=> '100mph'

7. В чем разница между методами экземпляра, класса и статическими методами в Python?

**Методы экземпляра**: принимают параметр self и относятся к определенному экземпляру класса.  
  
**Статические методы**: используют декоратор @staticmethod, не связаны с конкретным экземпляром и являются автономными (атрибуты класса или экземпляра не изменяются).  
  
**Методы класса**: принимают параметр cls, можно изменить сам класс.  
  
Проиллюстрируем разницу на вымышленном классе CoffeeShop:

**class** CoffeeShop:  
 specialty = 'espresso'  
  
 **def** \_\_init\_\_(self, coffee\_price):  
 self.coffee\_price = coffee\_price  
   
 # instance method  
 **def** make\_coffee(self):  
 print(f'Making {self.specialty} for ${self.coffee\_price}')  
   
 # static method   
 @staticmethod  
 **def** check\_weather():  
 print('Its sunny')  
  
 # class method  
 @classmethod  
 **def** change\_specialty(cls, specialty):  
 cls.specialty = specialty  
 print(f'Specialty changed to {specialty}')

У класса CoffeeShop есть атрибут specialty (фирменный напиток), установленный по умолчанию в значение 'espresso'. Каждый экземпляр CoffeeShop инициализируется с атрибутом coffee\_price. У него также три метода: метод экземпляра, статический метод и метод класса.  
  
Давайте инициализируем экземпляр с атрибутом coffee\_price, равным 5. Затем вызовем метод экземпляра make\_coffee:

coffee\_shop = CoffeeShop('5')  
coffee\_shop.make\_coffee()  
#=> Making espresso for $5

Теперь вызовем статический метод. Статические методы не могут изменять состояние класса или экземпляра, поэтому обычно используются для служебных функций, например, сложения двух чисел. Наши проверяют погоду. Говорят, что солнечно. Отлично!

coffee\_shop.check\_weather()  
#=> Its sunny

Теперь используем метод класса для изменения фирменного напитка (specialty), а затем сделаем кофе (make\_coffee):

coffee\_shop.change\_specialty('drip coffee')  
#=> Specialty changed to drip coffee  
  
coffee\_shop.make\_coffee()  
#=> Making drip coffee for $5

Обратите внимание, что make\_coffee раньше делал эспрессо, а теперь заваривает капельную кофеварку (drip coffee).

8. В чем разница между func и func()?

Вопрос должен проверить ваше понимание, что все функции в Python также являются объектами:

**def** func():  
 print('Im a function')  
  
func  
#=> function \_\_main\_\_.func>  
func()   
#=> Im a function

func — это представляющий функцию объект, который можно назначить переменной или передать другой функции. Функция func() с круглыми скобками вызывает функцию и возвращает результат.

9. Объясните, как работает функция map

Она возвращает объект (итератор), который перебирает значения, применяя функцию к каждому элементу. В случае необходимости объект можно преобразовать в список:

**def** add\_three(x):  
 **return** x + 3  
li = [1,2,3]  
list(map(add\_three, li))  
#=> [4, 5, 6]

Здесь к каждому элементу в списке мы добавляем число 3.

10. Объясните, как работает функция reduce

Это может быть сложновато сразу понять, пока вы не используете ее несколько раз.  
  
reduce принимает функцию и последовательность — и проходит по этой последовательности. На каждой итерации в функцию передаются как текущий элемент, так и выходные данные предыдущего элемента. В конце концов, возвращается одно значение:

**from** functools **import** reduce  
**def** add\_three(x,y):  
 **return** x + y  
li = [1,2,3,5]  
reduce(add\_three, li)  
#=> 11

Возвращается 11, что является суммой 1+2+3+5.

11. Объясните, как работает функция filter

Функция делает буквально то, о чем говорит ее название: она фильтрует элементы в последовательности.  
  
Каждый элемент передается функции, которая включает его в последовательность, если по условию получает True, и отбрасывает в случае False:

**def** add\_three(x):  
 **if** x % 2 == 0:  
 **return** **True**   
 **else**:  
 **return** **False**  
  
li = [1,2,3,4,5,6,7,8]  
  
[i **for** i **in** filter(add\_three, li)]  
#=> [2, 4, 6, 8]

Обратите внимание, как удалены все элементы, которые не делятся на 2.

12. Переменные в Python передаются по ссылке или по значению?

Будьте готовы спуститься в кроличью нору семантики, если загуглите этот вопрос и прочтете несколько первых страниц.  
  
В общем, все имена передаются по ссылке, но в некоторых ячейках памяти хранятся объекты, а в других — указатели на другие ячейки памяти.

name = 'object'

Давайте посмотрим, как это работает со строками. Создадим экземпляр имени и объекта, на который указывают другие имена. Затем удалим первое:

x = 'some text'  
y = x  
x **is** y #=> True  
  
**del** x # удаляем имя 'a' , но не объект в памяти  
  
z = y  
y **is** z #=> True

Мы видим, что все имена указывают на один и тот же объект в памяти, который остался нетронутым после операции удаления имени del x.  
  
Вот еще один интересный пример с функцией:

name = 'text'  
**def** add\_chars(str1):  
 print( id(str1) ) #=> 4353702856  
 print( id(name) ) #=> 4353702856  
   
 # новое имя, тот же объект  
 str2 = str1  
   
 # создаем новое имя (не отличается от предыдущего) и новый объект  
 str1 += 's'   
 print( id(str1) ) #=> 4387143328  
   
 # объект не изменился  
 print( id(str2) ) #=> 4353702856  
   
add\_chars(name)  
print(name) #=>text

Обратите внимание, что добавление буквы s в строку внутри функции создает новое имя — и новый объект тоже. Даже если у нового объекта то же самое имя, что и у существующего.

13. Как развернуть список?

Обратите внимание, что reverse() вызывается в списке и изменяет его. Сама функция не возвращает измененный список:

li = ['a','b','c']  
  
print(li)  
li.reverse()  
print(li)  
#=> ['a', 'b', 'c']  
#=> ['c', 'b', 'a']

14. Как работает умножение строк?

Посмотрим результат умножения строки 'cat' на 3:

'cat' \* 3  
#=> 'catcatcat'

В результате содержимое строки повторяется трижды.

15. Как работает умножение списка?

Посмотрим на результат умножения списка [1,2,3] на 2:

[1,2,3] \* 2  
#=> [1, 2, 3, 1, 2, 3]

Содержание списка [1,2,3] повторяется дважды.

16. Что означает self в классе?

Self ссылается на экземпляр класса. Так метод может обновлять объект, к которому принадлежит.  
  
Ниже передача self в \_\_init\_\_() дает возможность установить цвет экземпляра при инициализации:

**class** Shirt:  
 **def** \_\_init\_\_(self, color):  
 self.color = color  
   
s = Shirt('yellow')  
s.color  
#=> 'yellow'

17. Как объединить списки в Python?

Списки объединяются при сложении. Обратите внимание, что с массивами так не получается:

a = [1,2]  
b = [3,4,5]  
  
a + b  
#=> [1, 2, 3, 4, 5]

18. В чем разница между глубокой и мелкой копиями?

Обсудим это в контексте изменяемого объекта — списка. Для неизменяемых объектов глубокое и мелкое (поверхностное) копирование обычно не отличаются.  
  
Рассмотрим три сценария.  
  
**I) Поставьте ссылку на исходный объект**. Она отсылает новое имя li2 к тому же месту в памяти, на которое указывает li1. Поэтому любое изменение в li1 также происходит с li2:

li1 = [['a'],['b'],['c']]  
li2 = li1  
  
li1.append(['d'])  
print(li2)  
#=> [['a'], ['b'], ['c'], ['d']]

**II) Создайте мелкую копию оригинала**. Ее можно создать с помощью конструктора list() или mylist.copy().  
  
Мелкая копия создает новый объект, но заполняет его ссылками на оригинал. Таким образом, добавление нового объекта в исходный список li3 не отразится в li4, а вот изменение объектов в li3 — отразится:

li3 = [['a'],['b'],['c']]  
li4 = list(li3)  
  
li3.append([4])  
print(li4)  
#=> [['a'], ['b'], ['c']]  
  
li3[0][0] = ['X']  
print(li4)  
#=> [[['X']], ['b'], ['c']]

**III) Создайте глубокую копию**. Это делается с помощью copy.deepcopy(). Оригинал и копия полностью независимы, а изменения в одном не оказывают никакого влияния на другой:

**import** copy  
  
li5 = [['a'],['b'],['c']]  
li6 = copy.deepcopy(li5)  
  
li5.append([4])  
li5[0][0] = ['X']  
print(li6)  
#=> [['a'], ['b'], ['c']]

19. В чем разница между списками и массивами?

**Примечание**: в стандартной библиотеке Python есть объект array, но здесь мы специально обсуждаем массив из популярной библиотеки Numpy.  
  
Списки в каждом индексе можно заполнять разными типами данных. Массивы требуют однородных элементов.  
  
Арифметические действия в списках добавляют или удаляют элементы из списка. Арифметические действия на массивах соответствуют функциям линейной алгебры.  
  
Массивы используют меньше памяти и обладают значительно большей функциональностью.

20. Как объединить два массива?

Помните, что массивы — это не списки. Это библиотека Numpy и здесь работает линейная алгебра.  
  
Для объединения массивов нужно использовать соответствующую функцию Numpy:

**import** numpy **as** np  
  
a = np.array([1,2,3])  
b = np.array([4,5,6])  
  
np.concatenate((a,b))  
#=> array([1, 2, 3, 4, 5, 6])

21. Что вам нравится в Python?

**Примечание**: это очень субъективный вопрос, и логично адаптировать ответ в зависимости от того, на какую должность вы претендуете.  
  
Python очень удобочитаем, и есть так называемый «питоновский способ» решения почти любой задачи, то есть самый понятный, ясный и лаконичный код.  
  
Это мне кажется противоположностью Ruby, где часто много способов решить задачу без четких указаний, какой вариант предпочтительнее.

22. Какая ваша любимая библиотека в Python?

**Примечание**: это тоже субъективно, см. вопрос 21.  
  
При работе с большим количеством данных трудно найти что-то полезнее, чем pandas. С этой библиотекой обработка и визуализация данных становятся проще простого.

23. Назовите изменяемые и неизменяемые объекты

Неизменяемость означает, что состояние нельзя изменить после создания. Примеры: int, float, bool, string и tuple.  
  
Состояние изменяемых объектов можно изменить. Примеры: list, dict и set.

24. Как округлить число до трех десятичных знаков?

Используйте функцию round(value, decimal\_places):

a = 5.12345  
round(a,3)  
#=> 5.123

25. Как разбить список?

Синтаксис функции включает три аргумента: list[start:stop:step], где step — это интервал, через который возвращаются элементы:

a = [0,1,2,3,4,5,6,7,8,9]  
  
print(a[:2])  
#=> [0, 1]  
  
print(a[8:])  
#=> [8, 9]  
  
print(a[2:8])  
#=> [2, 3, 4, 5, 6, 7]  
  
print(a[2:8:2])  
#=> [2, 4, 6]

26. Что такое pickle?

Pickle — это модуль сериализации и десериализации объектов в Python.  
  
В примере ниже мы сериализуем и десериализуем список словарей:

**import** pickle  
  
obj = [  
 {'id':1, 'name':'Stuffy'},  
 {'id':2, 'name': 'Fluffy'}  
]  
  
**with** open('file.p', 'wb') **as** f:  
 pickle.dump(obj, f)  
  
**with** open('file.p', 'rb') **as** f:  
 loaded\_obj = pickle.load(f)  
  
print(loaded\_obj)  
#=> [{'id': 1, 'name': 'Stuffy'}, {'id': 2, 'name': 'Fluffy'}]

27. Какая разница между словарями и JSON?

Dict (словарь) — это тип данных Python, представляющий собой набор индексированных, но неупорядоченных пар ключ-значение.  
  
JSON — просто строка, которая следует заданному формату и предназначена для передачи данных.

28. Какие ORM вы использовали в Python?

Технология ORM (object-relational mapping, объектно-реляционное отображение) связывает модели данных (обычно в приложении) с таблицами БД и упрощает транзакции с базой данных.  
  
В контексте Flask обычно используется SQLAlchemy, а у Django собственная ORM.

29. Как работают any() и all()?

**Any** возвращает true, если хоть один элемент в последовательности соответствует условию, то есть является true.  
  
**All** возвращает true только в том случае, если условию соответствуют все элементы в последовательности.

a = [**False**, **False**, **False**]  
b = [**True**, **False**, **False**]  
c = [**True**, **True**, **True**]  
  
print( any(a) )  
print( any(b) )  
print( any(c) )  
#=> False  
#=> True  
#=> True  
  
print( all(a) )  
print( all(b) )  
print( all(c) )  
#=> False  
#=> False  
#=> True

30. Где быстрее поиск: в словарях или списках?

Поиск значения в списке занимает O(n) времени, потому что нужно пройти весь список.  
  
Поиск ключа в словаре занимает O(1) времени, потому что это хэш-таблица.  
  
Разница во времени может быть огромной, если значений много, поэтому для производительности обычно рекомендуют словари. Но у них есть другие ограничения, такие как необходимость уникальных ключей.

31. В чем разница между модулем и пакетом?

Модуль — это файл или набор файлов, которые импортируются вместе:

**import** sklearn

Пакет — это каталог с модулями:

**from** sklearn **import** cross\_validation

Таким образом, пакеты — это модули, но не все модули являются пакетами.

32. Как увеличить и уменьшить целое число в Python?

Инкремент и декремент можно сделать с помощью += и -=:

value = 5  
  
value += 1  
print(value)  
#=> 6  
  
value -= 1  
value -= 1  
print(value)  
#=> 4

33. Как вернуть двоичный код целого числа?

Используйте функцию bin():

bin(5)  
#=> '0b101'

34. Как удалить из списка дубликаты?

Это можно сделать путем преобразования списка в набор, а затем обратно в список:

a = [1,1,1,2,3]  
a = list(set(a))  
print(a)  
#=> [1, 2, 3]

Обратите внимание, что наборы не обязательно поддерживают порядок следования списка.

35. Как проверить, существует ли значение в списке?

Используйте in:

'a' **in** ['a','b','c']  
#=> True  
  
'a' **in** [1,2,3]  
#=> False

36. В чем разница между append и extend?

append добавляет значения в список, а extend добавляет в список значения из другого списка:

a = [1,2,3]  
b = [1,2,3]  
  
a.append(6)  
print(a)  
#=> [1, 2, 3, 6]  
  
b.extend([4,5])  
print(b)  
#=> [1, 2, 3, 4, 5]

37. Как получить абсолютное значение целого числа?

Это можно сделать с помощью функции abs():

abs(2)  
#=> 2  
  
abs(-2)  
#=> 2

38. Как объединить два списка в список кортежей?

Для объединения в список кортежей можно использовать функцию zip, причем не только двух, но трех и более списков.

a = ['a','b','c']  
b = [1,2,3]  
  
[(k,v) **for** k,v **in** zip(a,b)]  
#=> [('a', 1), ('b', 2), ('c', 3)]

39. Как отсортировать словарь по ключам, в алфавитном порядке?

Нельзя «отсортировать» словарь, поскольку словари не поддерживают упорядочение, но можно вернуть отсортированный список кортежей с ключами и значениями из словаря:

d = {'c':3, 'd':4, 'b':2, 'a':1}  
  
sorted(d.items())  
#=> [('a', 1), ('b', 2), ('c', 3), ('d', 4)]

40. Как реализуется наследование классов в Python?

В приведенном ниже примере класс Audi является наследником Car. И вместе с этим наследуются методы экземпляра родительского класса:

**class** Car():  
 **def** drive(self):  
 print('vroom')  
**class** Audi(Car):  
 **pass**  
audi = Audi()  
audi.drive()

41. Как удалить все пробелы из строки?

Можно разделить строку в местах пробелов, а затем снова соединить без пробелов:

s = 'A string with white space'  
  
''.join(s.split())  
#=> 'Astringwithwhitespace'

Двое читателей рекомендовали более каноничный способ замены, который следует духу Python, что «явное лучше, чем неявное». Он также быстрее работает, потому что здесь не создается новый объект списка:

s = 'A string with white space'  
s.replace(' ', '')  
#=> 'Astringwithwhitespace'

42. Почему мы используем enumerate() при итерации последовательности?

enumerate() позволяет отслеживать индекс при итерации последовательности. Это более нативный способ, чем определение и приращение целого числа, представляющего индекс:

li = ['a','b','c','d','e']  
  
**for** idx,val **in** enumerate(li):  
print(idx, val)  
#=> 0 a  
#=> 1 b  
#=> 2 c  
#=> 3 d  
#=> 4 e

43. В чем разница между pass, continue и break?

Заглушка **pass** означает «ничего не делать». Обычно мы используем эту функцию, потому что Python не позволяет создавать класс, функцию или оператор if без кода внутри.  
  
В приведенном ниже примере вылетит ошибка, если внутри i > 3 не будет кода, поэтому мы используем pass:

a = [1,2,3,4,5]  
**for** i **in** a:  
 **if** i > 3:  
 **pass**  
 print(i)  
#=> 1  
#=> 2  
#=> 3  
#=> 4  
#=> 5

**continue** отправляет вас к следующему элементу в цикле, останавливая выполнение для текущего элемента. Таким образом, print(i) никогда не получает значения i < 3:

**for** i **in** a:  
 **if** i < 3:  
 **continue**  
 print(i)  
#=> 3  
#=> 4  
#=> 5

**break** прерывает цикл, и последовательность больше не повторяется. Таким образом, на цифре 3 цикл прерывается, а этот и следующие элементы не печатаются:

**for** i **in** a:  
 **if** i == 3:  
 **break**  
 print(i)   
#=> 1  
#=> 2

44. Преобразуйте следующий цикл for в генератор списков (list comprehension)

Дан следующий цикл for:

a = [1,2,3,4,5]  
  
a2 = []  
**for** i **in** a:  
 a2.append(i + 1)  
print(a2)  
#=> [2, 3, 4, 5, 6]

Результат:

a3 = [i+1 **for** i **in** a]  
  
print(a3)  
#=> [2, 3, 4, 5, 6]

Генератор списка обычно считается более каноническим способом в Python, если он остается понятным.

45. Приведите пример тернарного оператора

Тернарный (условный) оператор — это однострочный оператор if/else.  
  
Синтаксис такой: a if condition else b.

x = 5  
y = 10  
  
'greater' **if** x > 6 **else** 'less'  
#=> 'less'  
  
'greater' **if** y > 6 **else** 'less'  
#=> 'greater'

46. Проверьте, что в строке только числа

Можно использовать isnumeric():

'123a'.isnumeric()  
#=> False  
  
'123'.isnumeric()  
#=> True

47. Проверьте, что в строке только буквы

Можно использовать isalpha():

'123a'.isalpha()  
#=> False  
  
'a'.isalpha()  
#=> True

48. Проверьте, что в строке только буквы и цифры

Здесь можно использовать isalnum():

'123abc...'.isalnum()  
#=> False  
  
'123abc'.isalnum()  
#=> True

49. Получите список ключей из словаря

Это можно сделать через передачу словаря в конструктор list():

d = {'id':7, 'name':'Shiba', 'color':'brown', 'speed':'very slow'}  
  
list(d)  
#=> ['id', 'name', 'color', 'speed']

50. Как перевести строку в верхний/нижний регистр?

Можно использовать строковые методы upper() и lower():

small\_word = 'potatocake'  
big\_word = 'FISHCAKE'  
  
small\_word.upper()  
#=> 'POTATOCAKE'  
  
big\_word.lower()  
#=> 'fishcake'

51. В чем разница между remove, del и pop?

**remove()** удаляет первое совпадающее значение:

li = ['a','b','c','d']  
  
li.remove('b')  
li  
#=> ['a', 'c', 'd']

**del** удаляет элемент по его индексу:

li = ['a','b','c','d']  
  
**del** li[0]  
li  
#=> ['b', 'c', 'd']

**pop()** удаляет элемент по индексу и возвращает этот элемент:

li = ['a','b','c','d']  
  
li.pop(2)  
#=> 'c'  
  
li  
#=> ['a', 'b', 'd']

52. Приведите пример генератора словарей (dict comprehension)

Ниже мы создадим словарь с буквами алфавита в качестве ключей и индексами в качестве значений:

# создаем список букв  
**import** string  
list(string.ascii\_lowercase)  
alphabet = list(string.ascii\_lowercase)  
  
# генерация словаря  
d = {val:idx **for** idx,val **in** enumerate(alphabet)}  
  
d  
#=> {'a': 0,  
#=> 'b': 1,  
#=> 'c': 2,  
#=> ...  
#=> 'x': 23,  
#=> 'y': 24,  
#=> 'z': 25}

53. Как выполняется обработка исключений в Python?

Для обработки исключений Python предоставляет конструкцию из трех слов: try, except и finally.  
  
Синтаксис выглядит примерно так:

**try**:  
 # попробовать сделать это  
**except**:  
 # если блок try не сработал, попробовать это  
**finally**:  
 # всегда делать это

Ниже упрощенный пример такой конструкции. Здесь блок try терпит неудачу, поскольку мы не можем складывать целые числа со строками. Блок except устанавливает val = 10, а затем блок finally выводит complete:

**try**:  
 val = 1 + 'A'  
**except**:  
 val = 10  
**finally**:  
 print('complete')  
   
print(val)  
#=> complete  
#=> 10

Конечно, невозможно на 100% угадать, какие вопросы зададут на собеседовании. Лучший способ подготовиться — программировать и еще раз программировать, накапливая опыт.  
  
Однако приведенный список точно поможет тем, кто готовится к собеседованию на позицию дата-сайентиста или junior/middle-разработчика Python.