Black Cow interview question - Answers

1. Explain this code:

a, \*\_, b = config( \*\*{

k:int(v)

for k,v in self.request.GET.items()

if k in self.keys\_allowed

} )

In this code, a and b represent the first and last items in a returned list of values. \_ signifies a value we are not interested in, \*\_ signifies there are multiple values we are not interested, those values being the values returned after ‘a’ but before the final value ‘b’. If the returned list for example was [21, 2001, 6, 20] and we decided to print a, \_, and b it would look like:

a = 21

\_ = [2001, 6]

b = 20

We’re assigning the values of a, \*\_, and b, by calling a function named config. To the config function we’re passing a dictionary of variable size which is only returning the keys and values from the returned GET request object if the keys are found in the scoped objects keys\_allowed property. The values for each of the accepted keys are converted to integers.

What happens inside the config function with this passed object is unclear, however the function returns a list.

Attached to the email with these answers is a python file demonstrating the behaviour of this code with a custom config function that returns the integer values for the keys as a list and prints the values in a, \*\_ (accessed as ‘\_’), and b.

1. Explain the following Python class and suggest some uses. How could we ensure that the correct datatypes are used?

class MyObject:

def \_\_init\_\_(self, v):

self.\_\_v = v

def \_\_lt\_\_(self, o):

return self.\_\_v < o.\_\_v

def \_\_eq\_\_(self, o):

return self.\_\_v == o.\_\_v

def \_\_repr\_\_(self):

return f"MyObject({self.\_\_v})"

def \_\_str\_\_(self):

return str(self.\_\_v)

**Classes**

This class is defined with some built in functionality for handling numerical values. Classes are used whenever you want reusable code and to implement encapsulation. Code reusability in the context of classes means creating instances of the same object type with different values without having to hardcode those objects individually throughout your application. The class itself is like a blueprint, and the instance is the representation of a real-world object. The class tells the program how to create the object and what functionality should be available to that object. The object in the example has several built-in functions which are being overridden. These are referred to as magic methods or special methods.

**Init**

The class has an init function which assigns a variable v which is passed to the object on instantiation and assigns it to ‘self.\_\_v’.

By appending a double underscore to the name of the variable v in the context of a class we are indicating that the property should be considered a private variable. Because python doesn’t enforce private/protected variables, it’s a best practice to let other developers working on the same codebase know that the variable is private. It lets other developers know the variable shouldn’t be accessed or changed from code outside the class.

**lt**

This special function overrides the less than operator for the object.

For example if you were to pass object\_a < object\_b and both objects were of the same class, then the current functionality would return a Boolean value telling you whether object\_a.\_\_v is smaller than object\_b.\_\_v. See section on type enforcement below to see how this customisation of existing operators may be useful.

**eq**

This special function overrides the == operator for the object. For example if you were to pass object\_a == object\_b and both objects were of the same class, it would compare the .\_\_v parameter for each of the objects and return a Boolean value.

**repr**

the \_\_repr\_\_ magic/special method is designed to return an unambiguous string representation of the object. This method can have a return-type type hint added to let you know a string is expected to be returned. Though we’re already using a formatted string so I doubt this would be confusing to even a junior developer.

**str**

There is an issue with the \_\_str\_\_ function here, that issue is that if we are going by the types we want in order to perform the mathematical operators on the parameter \_\_v then we need to convert it to a string before returning it from the function.

\_\_str\_\_ is called whenever you print the object without calling any other functions or operators on it. For example, print(obj\_a) would print whatever returned in the \_\_str\_\_ function.

**Type enforcement**

When it comes to typing, we can enforce the typing which we assign using type hints by using the isinstance standard python function and pythons built in TypeError with a custom error message. See the code file attached for how I’ve used type hints and isinstance to enforce typing.

1. Explain how we accomplish this so that our data cannot become inconsistent.

There are a few steps to consider when handling the data defined in the question and meeting the requirements for data consistency. The first suggestion I’d make is to complete all of the actions in a single SQL transaction, the reason you’d do this is to enforce an all-or-none set of changes to the database, if one of the operations fails, then none of the changes will be persisted. This helps for data consistency.

The second suggestion I’d make (because orders are being processed by multiple concurrent processes) is to implement row-level locking during this transaction as well as controlling the isolation level in the DBMS. Using Repeatable Read and Read Committed (at least in POSTGRESQL) ensures that a snapshot of the database can be viewed whilst also locking the rows currently being written. Repeatable read allows for consistent viewing of the data and is ideal for reporting purposes. Read Committed shows the latest data, so if at the same time as the statement is made, another insert statement is happening and you then execute a second select statement in your read committed query, you’ll see the updated row count.

Implementing retry logic to your application is important because if your query to run the transaction is unsuccessful, you want that query to be run again to attempt to update the data correctly.

You’ll notice in the query I provide in the python file attached, I’ve assumed a version column exists in the data, this can be used to check whether the version has changed since selecting the row before committing any changes.