

I've asked ChatGPT, Claude and Gemini;

You are a candidate for a software engineer job at a big tech company. given the following code, can you explain to a recruiter how a candidate might use duck typing to infer the line of questioning in an interview?

```
from random import seed
from random import random
from random import choice

seed(1)

class Question:
    def __init__(self, question, evidence):
        self.question = question
        self.check = evidence

    def ask(self):
        print(self.question)

    def answer(self, user_input):
        if self.check == None:
            self.check = user_input
            return None
        elif callable(self.check):
            return self.check(user_input)
        return self.check == user_input

# https://innocenceproject.org/police-interviews-versus-interrogation-fairbanks-four-case-11

class Interview:
    """
    Interrogation and interview are not synonyms; they have very different purposes and e
    Interviews are used in an investigation to gather information objective facts by a
    and allowing the witness to supply the evidence.
    Police conduct interviews when they don't yet know the answers to the questions they are
    """
    answers = []

    def save_answer(self, answer):
        self.answers.append(answer)
        return None

    def question(self):
        return Question("where were you last night at 8:00", self.save_answer)

class Interrogation:
    """
    Interrogations, on the other hand, are designed to extract confessions where police already have
    concrete evidence connecting the suspect to the crime.
    Most officers are trained in specific interrogation techniques that are intended to be u
    seasoned adult criminals.
    Because interrogations are so coercive, there's a danger in using them, rather than an i
    to solve a crime:
    They can produce false confessions that blind officers to other objective evidence.
    """

    def question(self,):
        return Question("where were you last night at 8:00", "10 downing street")

questioning_techniques = (
```

```
Interrogation(),
Interview()
)

questioning_techniques_by_result = {
    # question is open ended there is no predetermined true or false answer. the answer is a
    type(Interview().question().answer("nanyabiznis")): type(Interview()),
    # there are correct and incorrect answers
    type(Interrogation().question().answer("nanyabiznis")): type(Interrogation()),
}

class LeetCodeInterview:
    """
    You will have a limited time for the coding question, so it will be important to finish
    If possible, have a mock interview with a friend to simulate the interview experience.
    Be able to discuss the big-O complexity of your approaches.
    Don't forget to brush up on your data structures like lists, arrays, hash tables, hash m
    queues, graphs, trees, heaps. Also sorts, searches, and traversals (BFS, DFS).
    Also review recursion and iterative approaches.
    It's important to write your solution correctly and in time, so use the language you are
    """
    questions = {
        "merge sort": "O(n lgn)",
        "quick sort": "O(n^2)",
    }
    failed = False

    def question(self):
        list_of_questions = list(self.questions.items())
        question = choice(list_of_questions) # randomly select a question
        return Question("what is the worst case for {}".format(question[0]), question[1].__e

class WritingSamplesInterview:
    """
    Our hiring process is a little different than most: we believe that the best basis for
    of an engineer is not through their performance on an arbitrary oral exam, but rather th
    That is, we choose to use interviews not to assess you as a stranger, but rather to more
    understand the complete engineer who your work indicates you to be.
    """
    questions = [
        """
        The ultimate measure of an engineer is our work.
        Please submit at least one work sample (and no more than three), providing links if/
        This should be work that best reflects you as an engineer --
        work that you are proud of or you feel is otherwise representative of who you aspire
        If this work is entirely proprietary, please describe it as fully as you can, provid
        """,
        """
        A significant challenge of engineering is dealing with a system when it doesn't,
        in fact, work correctly. When systems misbehave, engineers must flip their disposit
        instead of a creator of their own heaven and earth, they must become a scientist,
        attempting to reason about a foreign world. Please provide an analysis sample: a wri
        analysis of system misbehavior from some point in your career. If such an analysis i
        available (as it might not be if one's work has been strictly proprietary),
        please recount an incident in which you analyzed system misbehavior,
        including as much technical detail as you can recall.
        """,
    ]

```

```
]
answers = []

def save_answer(self, answer):
    self.answers.append(answer)
    return None

def question(self):
    return Question(choice(self.questions), self.save_answer)

class Candidate:
    the_one_thing_i_know = "O(n lgn)"
    line_of_questioning = type(None)

    def do(self, interview):
        q = interview.question()
        result = q.answer(self.the_one_thing_i_know)
        #
        # uncomment the line below to infer the line of questioning by question results
        #
        self.line_of_questioning = questioning_techniques_by_result[type(
            result)]

    def reflect(self):
        return self.line_of_questioning

interviews = {
    # https://www.facebook.com/careers/life/preparing-for-your-software-engineering-interview
    'facebook': LeetCodeInterview(),
    # https://docs.google.com/document/d/1Xt0fg-fMQfZoq8Y3oSAKjEgDQCRHx-GMSmPcxDEea4A/edit
    'oxide': WritingSamplesInterview(),
}

for interviewer in interviews:
    candidate = Candidate()
    interview = interviews[interviewer]
    candidate.do(interview)
    print(
        "{interview_type} quacks like a {interview_duck_type}".format(
            interview_duck_type=candidate.reflect(),
            interview_type=type(interview),
        )
    )
```

chatgpt

Certainly! Here's an explanation that highlights how duck typing is used in the given Python code to infer the line of questioning during an interview:

Explanation for Recruiter

In the provided code, duck typing is utilized to determine the nature of the interview questions a candidate might encounter. Duck typing in Python allows an object's suitability to be determined by the presence of certain methods and properties, rather than the object's actual type.

Code Overview

1. Classes for Different Types of Interviews:

-

Interview : Represents an open-ended interview where answers are saved for later analysis.

-

Interrogation : Represents a more directive interview where answers are compared against known facts.

-

LeetCodeInterview : Focuses on technical questions with specific correct answers.

-

WritingSamplesInterview : Requests detailed written work samples or analysis from the candidate.

2. Question Class:

- Manages the asking and answering of questions. The **answer** method can handle different types of verification:

- Saving the answer.
- Checking against a callable (function) for validation.
- Comparing with a predetermined answer.

3. Candidate Class:

- The **Candidate** class simulates a candidate taking the interview.
- The **do** method is where duck typing comes into play. It:
 - Retrieves a question from the interview.
 - Attempts to answer it with a known piece of information.
 - Uses the type of the result from the answer method to infer the interview type.

4. Duck Typing for Inferring Line of Questioning:

- The dictionary **questioning_techniques_by_result** maps the type of the result of answering a question to the type of interview (either **Interview** or **Interrogation**).
- When a candidate answers a question, the type of the result is checked against this dictionary to infer the interview technique used.

How Duck Typing is Used

- Flexible Answer Checking:

- The **answer** method in the **Question** class handles various ways to verify an answer. This flexibility is key to allowing different types of interviews to be managed without explicitly checking the type of interview.

- Type-Based Inference:

- By mapping the result of **answer** to interview types using the **questioning_techniques_by_result** dictionary, the **Candidate** class can infer the nature of the

interview dynamically.

- This approach uses the principle of "if it quacks like a duck, it's a duck" the candidate doesn't need to know the exact class of the interview, only how the response behaves.

Practical Implications for an Interview

- Dynamic Adaptability:

- The candidate class adapts dynamically to different interview styles based on the behavior of the answers rather than their explicit types.

- Efficient Handling of Multiple Interview Types:

- This method allows the same candidate logic to handle multiple types of interviews efficiently, showcasing the candidate's ability to manage diverse interview scenarios without hard-coding for each type.

In summary, the code leverages duck typing to allow a candidate to infer the type of interview they are undergoing based on the nature of the responses to questions. This flexible approach highlights the adaptability and robustness of the candidate's code in handling varying interview formats.

claude

Certainly. I'll explain how a candidate might use duck typing to infer the line of questioning in an interview based on the provided code. Let me break it down for you:

1. Understanding Duck Typing: Duck typing is a concept in programming where the type or class of an object is less important than the methods it defines. The idea is: "If it walks like a duck and quacks like a duck, then it must be a duck." In other words, we determine the nature of an object by its behavior rather than its explicit type.
2. Interview Types in the Code: The code defines different types of interviews, including **Interview**, **Interrogation**, **LeetCodeInterview**, and **WritingSamplesInterview**.
3. Inferring the Line of Questioning: The key to inferring the line of questioning lies in the **questioning_techniques_by_result** dictionary. This dictionary maps the type of result from a question to the type of interview technique:

```
questioning_techniques_by_result = {  
    type(Interview().question().answer("nanyabiznis")): type(Interview()),  
    type(Interrogation().question().answer("nanyabiznis")): type(Interrogation()),  
}
```

1. The Candidate's Approach: In the **Candidate** class, there's a method called **do**:

```
def do(self, interview):  
    q = interview.question()  
    result = q.answer(self.the_one_thing_i_know)  
    self.line_of_questioning = questioning_techniques_by_result[type(result)]
```

This method is using duck typing to infer the line of questioning. Here's how:

- a. It asks a question and provides an answer.
- b. It looks at the type of the result.
- c. Based on the type of the result, it determines the type of interview technique being used.

1. Why This Works:

- Different types of interviews handle answers differently.
 - An **Interview** saves all answers without judging them (open-ended).
 - An **Interrogation** compares the answer to a predetermined correct answer.
 - By observing how the answer is treated (saved vs. compared), the candidate can infer the type of interview.
2. The Reflection: The **reflect** method in the **Candidate** class returns the inferred line of questioning:
- ```
def reflect(self):
 return self.line_of_questioning
```

In essence, the candidate is "reverse-engineering" the interview process. By observing how their answers are handled (the "quack" of the duck), they're figuring out what kind of interview they're in (the "type" of duck), without explicitly being told.

This approach demonstrates a deep understanding of both the technical aspects of duck typing and the nuances of different interview styles, showcasing problem-solving skills that go beyond just answering the questions asked.

### **gemini**

**Absolutely! Here's how a software engineer candidate might explain the use of duck typing in this code to a recruiter, along with how it could help infer the line of questioning in a technical interview:**

### **Duck Typing Explained**

Duck typing is a programming concept where an object's suitability for a task is determined by its behavior (the methods it has) rather than its explicit type or class. The phrase "If it walks like a duck and quacks like a duck, then it must be a duck" perfectly captures this idea.

In the provided code, duck typing is subtly used to differentiate between different types of interviews and questioning techniques:

1. **Common Interface:** The **Question**, **Interview**, and **Interrogation** classes all share a **question()** method. This method returns a **Question** object.
2. **Differing Implementations:**
  - \*  
**Interview.question()** is designed for open-ended responses.
  - \*  
**Interrogation.question()** expects a specific, predetermined answer.
  - \*  
**LeetCodeInterview.question()** tests algorithm knowledge.
  - \*  
**WritingSamplesInterview.question()** requests work samples.
3. **Implicit Typing:** The code doesn't explicitly check if an object is an **Interview** or **Interrogation** result of calling **question().answer()** :

- \* If the answer is **None** , it infers an **Interview** (open-ended).
- \* If the answer is **True** or **False** , it infers an **Interrogation** (right/wrong).

### Inferring Line of Questioning

This duck typing strategy allows the **Candidate** class to dynamically adapt to the interview style:

1. **The do()** method:

- \* Calls **interview.question()** to get a question.
- \* Attempts to answer with the candidate's single known response ( **self.the\_one\_thing\_i\_know** ).

2. **The reflect()** method:

- \* Uses a dictionary ( **questioning\_techniques\_by\_result** ) to map the .I type of result from **answer()** to the corresponding interview type.
- \* Returns this inferred interview type.

### How a Candidate Would Explain This

"The code demonstrates duck typing by focusing on the behavior of the interview objects. The different interview types share a common interface ( **question()** ), but their implementations differ in how they handle answers. By examining the result of the **answer()** method, we can infer whether the interviewer is looking for open-ended responses (like in an **Interview** ), expecting specific answers (like in an **Interrogation** ), or testing algorithmic knowledge (like in a **LeetCodeInterview** ).

This approach simulates how a candidate might adapt their strategy in a real interview. If a question seems open-ended, they'd provide a thoughtful explanation. If the interviewer seems to be looking for a specific answer, the candidate might try to guess the correct response. In a technical interview, the candidate would focus on demonstrating their algorithmic skills."

### Benefits to the Company

- \* **Flexibility:** This approach allows the company to easily add new interview types without major code changes.
- \* **Adaptability:** The candidate's ability to infer the line of questioning showcases their problem-solving skills and adaptability in a dynamic interview environment.

**Important Note:** The code snippet is a simplified example. In a real interview, a candidate would use a much broader range of responses and strategies to determine the interviewer's focus.