Python Guide for the Data Domain

Data Types and Structures

Lists

Definition: Lists are ordered, mutable collections used to store multiple items.

Use Case: Used when you need to store a sequence of items, such as a series of values from a dataset.

```
Syntax:

list_name = [item1, item2, ...]
```

```
Example:



# Create a list of numbers

numbers = [1, 2, 3, 4, 5]

print(numbers[0])  # Access first item

numbers.append(6)  # Add an item

print(numbers)  # [1, 2, 3, 4, 5, 6]
```


Dictionaries

Definition: Dictionaries are collections of key-value pairs, used for fast lookups.

Use Case: Ideal for scenarios where data is labeled, such as storing attributes of an entity.

```
Syntax:

dict_name = {key1: value1, key2: value2, ...}
```

```
Example:



# Store employee data

employee = {"name": "John", "age": 30, "department": "HR"}

print(employee["name"])  # Access the value of a key

employee["age"] = 31  # Update value

print(employee)  # {'name': 'John', 'age': 31, 'department': 'HR'}
```

Tuples

Definition: Tuples are immutable, ordered collections.

Use Case: Used for fixed data that shouldn?t be altered, like coordinates or configurations.

```
Syntax:

tuple_name = (item1, item2, ...)
```

```
Example:



# Store immutable data

coordinates = (10, 20)

print(coordinates[0])  # Access first element
```

File Handling

CSV File Handling

Definition: CSV files store tabular data in plain text, separated by commas.

Use Case: Commonly used for data exchange between applications.

```
Syntax:


import csv
with open('file.csv', mode='r') as file:
    csv_reader = csv.reader(file)
```

```
Example:


import csv
# Reading a CSV file
with open('data.csv', mode='r') as file:
    reader = csv.reader(file)
    for row in reader:
        print(row)
```


JSON File Handling

Definition: JSON files store data in a structured, human-readable format.

Use Case: Useful for APIs and structured data handling.

```
Syntax:


import json
```

```
with open('file.json', mode='r') as file:

    data = json.load(file)
```

```
Example:


import json

# Reading a JSON file

with open('data.json', 'r') as file:

    data = json.load(file)

print(data['key'])
```

50 Python Interview Questions with Answers

Q: What are Python's key features?

A: Python is an interpreted, high-level, and dynamically typed programming language. It supports object-oriented, procedural, and functional programming paradigms. Python is known for its simplicity and readability.

Q: Explain the difference between a list and a tuple.

A: Lists are mutable, meaning their contents can be changed, while tuples are immutable. Lists are defined with square brackets `[]`, and tuples with parentheses `()`. Lists are typically used for homogeneous data, while tuples are used for heterogeneous data.

Q: How is memory managed in Python?

A: Python uses dynamic memory management handled by its memory manager. The garbage collector reclaims memory occupied by objects no longer in use.

Q: What is a Python decorator?

A: A decorator is a function that modifies the behavior of another function or method. Decorators are applied using the `@decorator_name` syntax.

Q: How does Python handle multithreading?

A: Python's Global Interpreter Lock (GIL) allows only one thread to execute at a time per process. For CPU-bound tasks, multiprocessing is preferred over multithreading in Python.

Q: What are Python's built-in data types?

A: Some of Python's built-in data types include: int, float, str, list, tuple, dict, set, frozenset, complex, bool, and NoneType.

Python Guide for the Data Domain

Q: Explain the concept of Python's duck typing.

A: Duck typing in Python means that an object's suitability is determined by the presence of certain methods and properties rather than the object's actual type. 'If it looks like a duck and quacks like a duck, it must be a duck.'

Q: How can you achieve inheritance in Python?

A: Inheritance in Python is achieved by defining a child class that derives properties and methods from a parent class. Syntax: `class ChildClass(ParentClass):`

Q: What is the difference between shallow copy and deep copy in Python?

A: A shallow copy creates a new object but inserts references to the original objects. A deep copy creates a new object and recursively copies all the objects within the original object.

Q: What is the purpose of Python's `with` statement?

A: The `with` statement is used for resource management, such as opening files. It ensures that resources are properly cleaned up after use.

Q: How can you handle exceptions in Python?

A: Exceptions in Python can be handled using the `try...except` block. Optionally, `finally` and `else` can be added for cleanup and post-exception code.

Q: What are Python's comprehensions?

A: Comprehensions are a concise way to construct collections like lists, sets, or dictionaries. Example: `[x**2 for x in range(10)]` creates a list of squares.

Q: What is Python's `None` type?

A: `None` is Python's null value, used to represent the absence of a value or a null object.

Python Guide for the Data Domain

Q: Explain Python's `lambda` functions.

A: Lambda functions are anonymous, single-expression functions defined with the `lambda` keyword. Example: `lambda x: x**2`.

Q: What is the difference between `is` and `==` in Python?

A: `is` checks for object identity (whether two references point to the same object), while `==` checks for value equality.

Q: How can you create a virtual environment in Python?

A: You can create a virtual environment using the `venv` module: `python -m venv env_name`.

Q: Explain the purpose of Python's `sys` module.

A: The `sys` module provides access to system-specific parameters and functions, like `sys.argv` for command-line arguments.

Q: What is the `__init__.py` file in Python?

A: `__init__.py` is used to mark a directory as a Python package. It can also include initialization code for the package.

Q: What is the `@staticmethod` decorator in Python?

A: The `@staticmethod` decorator allows you to define a method that doesn't require access to the class or instance.

Q: What is the difference between `break`, `continue`, and `pass` in Python?

A: `break` exits a loop, `continue` skips to the next iteration, and `pass` is a no-op used as a placeholder.