Python FAQ PDF 2: Intermediate Python Topics (Long Answers)

Q1: What are list comprehensions and why are they useful?
A: List comprehensions provide a concise way to create lists in Python. They consist of brackets containing an expression followed by a for clause, and optionally, if clauses. List comprehensions are more readable and often faster than using traditional for loops for generating lists. For example, [x*x for x in range(10) if x%2==0] creates a list of squares of even numbers from 0 to 9.

Q2: How does Python support object-oriented programming (OOP)?
A: Python fully supports OOP by allowing the creation and use of classes and objects. It supports inheritance, encapsulation, and polymorphism. Classes are defined using the class keyword, and methods are defined within classes. Python also supports special methods (like __init__ for constructors) and allows operator overloading. OOP in Python helps in modeling complex systems and code reuse.

Q3: What are generators and how do they differ from normal functions?
A: Generators are functions that use the yield statement to return values one at a time, suspending and resuming their state between calls. Unlike normal functions that return all results at once, generators are memory-efficient and suitable for handling large datasets or infinite sequences, as values are produced on demand.

Q4: What is a lambda function in Python?
A: A lambda function is a small anonymous function defined with the lambda keyword. It can have any number of arguments but only one expression. Lambda functions are often used for short, throwaway functions, especially as arguments to higher-order functions like map(), filter(), and sorted().

Q5: Explain the concept of mutability and immutability in Python.
A: Mutable objects can be changed after creation (like lists, dictionaries, and sets), while immutable objects cannot (like tuples, strings, and integers). Understanding mutability is important for avoiding unexpected behavior, especially when passing objects as function arguments or using them as dictionary keys.

Q6: What are Python's built-in functions and how can you use them?
A: Python comes with many built-in functions such as len(), sum(), map(), filter(), zip(), and enumerate(). They provide common functionality without needing to import additional modules. Knowing and leveraging these built-ins can make code more concise, readable, and efficient.

Q7: How does Python handle file input and output (I/O)?
A: Python handles file I/O with the built-in open() function, which returns a file object. You can read from or write to files using methods like read(), readline(), write(), and writelines().

Python supports both text and binary file modes and encourages the use of the with statement for automatic resource management.

Q8: What is the difference between shallow copy and deep copy in Python?
A: A shallow copy creates a new object but does not recursively copy objects contained within the original object; instead, it copies references to those objects. A deep copy creates a new object and recursively copies all nested objects. Shallow copies can be made using the copy() method, while deep copies require the copy module's deepcopy() function.

Q9: How does Python's import system work?
A: Python's import system allows you to include reusable code from modules and packages. The import statement loads a module and makes its functions, classes, and variables accessible. Python searches for modules in directories listed in sys.path, including the current directory, installed packages, and standard library.

Q10: What are virtual environments and why should you use them?
A: Virtual environments are isolated Python environments that allow you to manage dependencies for different projects independently. Tools like venv and virtualenv create separate environments with their own Python interpreter and package directories. This prevents conflicts between project dependencies and makes deployment easier.