Python Concepts and Applications

1) What is Garbage Collection in Python?

Garbage collection in Python refers to the process of freeing up memory by deleting objects that are no longer in use. Python uses two techniques for memory management: reference counting and cyclic garbage collection.  
- Reference Counting: Each object in Python maintains a count of references to it. When the reference count drops to zero, the object is deallocated.  
- Garbage Collection: In addition to reference counting, Python has a garbage collector to detect and clean up circular references (where objects reference each other).

2) Key Differences Between NumPy Arrays and Python Lists

The main differences between NumPy arrays and Python lists are:  
- Homogeneity: NumPy arrays are homogeneous, meaning all elements must be of the same type, while Python lists can store elements of different types.  
- Memory Efficiency: NumPy arrays store data more efficiently in memory because they are stored in contiguous memory blocks, whereas Python lists are stored as objects with pointers to their elements.  
- Performance: NumPy arrays are faster for numerical computations because they allow element-wise operations and make use of vectorization, avoiding Python’s slow loops.

3) List Comprehension in Python

List comprehension provides a concise way to create lists based on existing lists or conditions. It allows for generating a list from an expression followed by a `for` loop.  
Example: Generate a list of squared values:

```python  
squares = [x\*\*2 for x in range(10)]  
print(squares)  
# Output: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]  
```

You can also use list comprehension to filter elements based on a condition. Example: Filter even numbers:

```python  
numbers = [x for x in range(10) if x % 2 == 0]  
print(numbers)  
# Output: [0, 2, 4, 6, 8]  
```

4) Shallow vs. Deep Copying in Python

In Python, copying objects can be done in two ways: shallow and deep copy.  
- Shallow Copy: Creates a new object, but does not create copies of objects contained within the original. Instead, it copies references to the original objects. This means changes to mutable objects within the copy will affect the original object.

- Deep Copy: Creates a new object and recursively copies all objects inside it. Changes to the deep copy won't affect the original object. Deep copying is implemented using the `copy` module's `deepcopy()` function.

Example:  
```python  
import copy  
original\_list = [[1, 2, 3], [4, 5, 6]]  
shallow = copy.copy(original\_list)  
deep = copy.deepcopy(original\_list)  
```  
Modifying shallow copy affects the original, while deep copy does not.

5) Difference Between Lists and Tuples in Python

Lists and tuples are both sequence types in Python, but they have key differences:  
- Mutability: Lists are mutable, meaning their contents can be changed after creation. Tuples are immutable, and once created, their contents cannot be modified.  
- Performance: Tuples are faster than lists for lookups, and they use less memory because of their immutability.

Example:  
```python  
my\_list = [1, 2, 3]  
my\_tuple = (1, 2, 3)  
my\_list[0] = 10 # Works  
my\_tuple[0] = 10 # Error  
```