Question 1.1: Python Concepts and Examples

1. **Difference between static and dynamic variables in Python:**
    - **Static variables:** These are variables that are shared among all instances of a class. They are defined within a class but outside any instance methods.
    - **Dynamic variables:** These are variables that are defined at runtime. They can be modified during execution.

```
class MyClass:
    static_var = 0  # Static variable

    def __init__(self, dynamic_var):
        self.dynamic_var = dynamic_var  # Dynamic variable

obj1 = MyClass(1)
obj2 = MyClass(2)
print(obj1.static_var, obj1.dynamic_var)
print(obj2.static_var, obj2.dynamic_var)
```

2. **Purpose of `pop`, `popitem`, and `clear()` in a dictionary:**
    - `pop(key)`: Removes the item with the specified key and returns its value.
    - `popitem()`: Removes and returns the last inserted key-value pair.
    - `clear()`: Removes all items from the dictionary.

```
my_dict = {'a': 1, 'b': 2, 'c': 3}
print(my_dict.pop('b'))  # Output: 2
print(my_dict.popitem())  # Output: ('c', 3)
my_dict.clear()
print(my_dict)  # Output: {}
```

3. **FrozenSet:**
    - A `frozenset` is an immutable version of a set. It is hashable and can be used as a key in dictionaries or stored in other sets.

```
fs = frozenset([1, 2, 3, 4])
print(fs)
```

4. **Difference between mutable and immutable data types:**
    - **Mutable:** Can be changed after creation (e.g., list, dictionary).
    - **Immutable:** Cannot be changed after creation (e.g., tuple, string).

```
# Mutable example
my_list = [1, 2, 3]
my_list.append(4)

# Immutable example
my_tuple = (1, 2, 3)
```

5. **`__init__`:**
    - The constructor method in Python classes. It's called when an instance of the class is created.

python
Copy code

```
class MyClass:
    def __init__(self, value):
        self.value = value
obj = MyClass(10)
```

6. **Docstring in Python:**
    - A docstring is a string literal used to document a module, class, function, or method.

```
def my_function():

    """This is a docstring."""
    pass
```

7. **Unit tests:**
    - Unit tests are automated tests written and run to ensure that a section of an application (known as the "unit") meets its design and behaves as intended.
8. **Break, continue, and pass in Python:**
    - `break`: Exits the loop.
    - `continue`: Skips the rest of the code inside the loop for the current iteration.
    - `pass`: Does nothing; acts as a placeholder.

```
for i in range(5):
    if i == 2:
        break
    print(i)
```

9. **Use of `self` in Python:**
    - `self` represents the instance of the class. It allows access to the attributes and methods of the class.

10. **Global, protected, and private attributes:**
    - **Global:** Defined at the module level.
    - **Protected:** Indicated by a single underscore (_variable).
    - **Private:** Indicated by a double underscore (__variable).
11. **Modules and packages in Python:**
    - **Module:** A single file containing Python code.
    - **Package:** A directory containing multiple modules.
12. **Lists and tuples:**
    - **List:** Mutable, ordered collection.
    - **Tuple:** Immutable, ordered collection.
    - **Key difference:** Lists are mutable, tuples are immutable.

```
my_list = [1, 2, 3]
my_tuple = (1, 2, 3)
```

13. **Interpreted language & dynamically typed language:**
    - **Interpreted language:** Executes instructions directly without prior compilation.
    - **Dynamically typed language:** Type checking is performed at runtime.
14. **Dict and List comprehensions:**
    - A concise way to create dictionaries and lists.

```
my_list = [x for x in range(10)]
my_dict = {x: x**2 for x in range(10)}
```

15. **Decorators in Python:**
    - Functions that modify the behavior of another function.

```
def my_decorator(func):
    def wrapper():
        print("Something is happening before the function is
called.")
        func()
        print("Something is happening after the function is
called.")
    return wrapper
@my_decorator
def say_hello():
    print("Hello!")
say_hello()
```

16. **Memory management in Python:**
    - Managed by Python's memory manager and includes garbage collection.

17. **Lambda in Python:**
    - Anonymous function.

```
add = lambda x, y: x + y
```

18. **`split()` and `join()` functions in Python:**
    - `split()`: Splits a string into a list.
    - `join()`: Joins a list of strings into a single string.

```
s = "hello world"
print(s.split())  # ['hello', 'world']
print(" ".join(['hello', 'world']))  # 'hello world'
```

19. **Iterators, iterable & generators in Python:**
    - **Iterator:** Object with a `__next__()` method.
    - **Iterable:** Object with an `__iter__()` method.
    - **Generator:** Special type of iterator created using a function with `yield`.
20. **Difference between `xrange` and `range` in Python:**
    - `range()`: Returns a list in Python 2, an iterator in Python 3.
    - `xrange()`: Returns an iterator in Python 2 (not available in Python 3).
21. **Pillars of OOPs:**
    - Encapsulation, Abstraction, Inheritance, and Polymorphism.
22. **Checking if a class is a child of another class:**
    - Use `issubclass()` function.

```
class Parent:
    pass
class Child(Parent):
    pass
print(issubclass(Child, Parent))  # True
```

23. **Inheritance in Python:**
    - Allows one class to inherit attributes and methods from another class.

```
class Parent:

    def __init__(self, name):
        self.name = name

class Child(Parent):
    def __init__(self, name, age):
        super().__init__(name)
        self.age = age
```

24. **Encapsulation:**
    - Restricting access to some of an object's components

```
class MyClass:

    def __init__(self, value):
        self.__value = value  # Private attribute
    def get_value(self):
        return self.__value
```

25. **Polymorphism:**
    - Ability to use a common interface for multiple forms.

```
class Animal:

    def sound(self):
        pass
class Dog(Animal):
    def sound(self):
        return "Bark"
class Cat(Animal):
    def sound(self):

      return "Meow"
```