PW SKILLS

Assignmment

Object Oriented Programming

1. Explain the importance of Functions.
2. Write a basic function to greet students.
3. What is the difference between print and return statements?
4. What are *args and **kwargs?
5. Explain the iterator function.
6. Write a code that generates the squares of numbers from 1 to n using a generator.
7. Write a code that generates palindromic numbers up to n using a generator.
8. Write a code that generates even numbers from 2 to n using a generator.
9. Write a code that generates powers of two up to n using a generator.
10. Write a code that generates prime numbers up to n using a generator.
11. Write a code that uses a lambda function to calculate the sum of two numbers.
12. Write a code that uses a lambda function to calculate the square of a given number.
13. Write a code that uses a lambda function to check whether a given number is even or odd.
15. Write a code that uses a lambda function to concatenate two strings.
16. Write a code that uses a lambda function to find the maximum of three given numbers.
17. Write a code that generates the squares of even numbers from a given list.
18. Write a code that calculates the product of positive numbers from a given list.
19. Write a code that doubles the values of odd numbers from a given list.
20. Write a code that calculates the sum of cubes of numbers from a given list.
21. Write a code that filters out prime numbers from a given list.
22. Write a code that uses a lambda function to calculate the sum of two numbers.
23. Write a code that uses a lambda function to calculate the square of a given number.
24. Write a code that uses a lambda function to check whether a given number is even or odd.
25. Write a code that uses a lambda function to concatenate two strings.
26. Write a code that uses a lambda function to find the maximum of three given numbers.
27. What is encapsulation in OOP?

28. Explain the use of access modifiers in Python classes.
29. What is inheritance in OOP?
30. Define polymorphism in OOP.
31. Explain method overriding in Python.
32. Define a parent class Animal with a method make_sound that prints "Generic animal sound". Create a child class Dog inheriting from Animal with a method make_sound that prints "Woof!".
33. Define a method move in the Animal class that prints "Animal moves". Override the move method in the Dog class to print "Dog runs."
34. Create a class Mammal with a method reproduce that prints "Giving birth to live young." Create a class DogMammal inheriting from both Dog and Mammal.
35. Create a class GermanShepherd inheriting from Dog and override the make_sound method to print "Bark!"
36. Define constructors in both the Animal and Dog classes with different initialization parameters.
37. What is abstraction in Python? How is it implemented?
38. Explain the importance of abstraction in object-oriented programming.
39. How are abstract methods different from regular methods in Python?
40. How can you achieve abstraction using interfaces in Python?
41. Can you provide an example of how abstraction can be utilized to create a common interface for a group of related classes in Python?
42. How does Python achieve polymorphism through method overriding?
43. Define a base class with a method and a subclass that overrides the method.
44. Define a base class and multiple subclasses with overridden methods.
45. How does polymorphism improve code readability and reusability?
46. Describe how Python supports polymorphism with duck typing.
47. How do you achieve encapsulation in Python?
48. Can encapsulation be bypassed in Python? If so, how?
49. Implement a class BankAccount with a private balance attribute. Include methods to deposit, withdraw, and check the balance.
50. Develop a Person class with private attributes name and email, and methods to set and get the email.
51. Why is encapsulation considered a pillar of object-oriented programming (OOP)?

52. Create a decorator in Python that adds functionality to a simple function by printing a message before and after the function execution.

53. Modify the decorator to accept arguments and print the function name along with the message.

54. Create two decorators, and apply them to a single function. Ensure that they execute in the order they are applied.

55. Modify the decorator to accept and pass function arguments to the wrapped function.

56. Create a decorator that preserves the metadata of the original function.

57. Create a Python class `Calculator` with a static method `add` that takes in two numbers and returns their sum.

58. Create a Python class `Employee` with a class `method get_employee_count` that returns the total number of employees created.

59. Create a Python class `StringFormatter` with a static method `reverse_string` that takes a string as input and returns its reverse.

60. Create a Python class `Circle` with a class method `calculate_area` that calculates the area of a circle given its radius.

61. Create a Python class `TemperatureConverter` with a static method `celsius_to_fahrenheit` that converts Celsius to Fahrenheit.

62. What is the purpose of the __str__() method in Python classes? Provide an example.

63. How does the __len__() method work in Python? Provide an example.

64. Explain the usage of the __add__() method in Python classes. Provide an example.

65. What is the purpose of the __getitem__() method in Python? Provide an example.

66. Explain the usage of the __iter__() and __next__() methods in Python. Provide an example using iterators.

67. What is the purpose of a getter method in Python? Provide an example demonstrating the use of a getter method using property decorators.

68. Explain the role of setter methods in Python. Demonstrate how to use a setter method to modify a class attribute using property decorators.

69. What is the purpose of the @property decorator in Python? Provide an example illustrating its usage.

70. Explain the use of the @deleter decorator in Python property decorators. Provide a code example demonstrating its application.

71. How does encapsulation relate to property decorators in Python? Provide an example showcasing encapsulation using property decorators.