**Classwork 6**

**Classwork\_OOP\_a**

1)

class BasicPlan:  
 can\_stream = True  
 can\_download = True  
 has\_SD = True  
 has\_HD = False  
 has\_UHD = False  
 num\_of\_devices = 1  
 price = "$8.99"  
  
class StandardPlan(BasicPlan):  
 has\_HD = True  
 num\_of\_devices = 2  
 price = "$12.99"  
  
class PremiumPlan(StandardPlan):  
 has\_UHD = True  
 num\_of\_devices = 4  
 price = "$15.99"  
  
print(BasicPlan.has\_SD)  
print(PremiumPlan.has\_SD)  
print(BasicPlan.has\_UHD)  
print(BasicPlan.price)  
print(PremiumPlan.num\_of\_devices)

2)

import math  
class Circle:  
 def \_\_init\_\_(self, radius):  
 self.radius = radius  
 def getArea(self):  
 return math.pi \* self.radius \*\* 2  
 def getPerimeter(self):  
 return 2 \* math.pi \* self.radius  
circy = Circle(11)  
print(circy.getArea())  
print(circy.getPerimeter())

3)

class Person:  
 def \_\_init\_\_(self, name, age):  
 self.name = name  
 self.age = age  
 def compare\_age(self,other):  
 if self.age > other.age:  
 return f"{other.name} is younger than {self.name}"  
 elif self.age == other.age:  
 return f"{other.name} is the same age as {self.name}"  
 else:  
 return f"{other.name} is older than {self.name}"  
p1 = Person("Samuel", 24)  
p2 = Person("Joel", 36)  
p3 = Person("Lily", 24)  
print(p1.compare\_age(p2))  
print(p2.compare\_age(p1))  
print(p1.compare\_age(p3))

4)

class Calculator:  
 def \_\_init\_\_(self):  
 pass  
 def add(self, a, b):  
 return a + b  
 def subtract(self, a, b):  
 return a - b  
 def multiply(self, a, b):  
 return a \* b  
 def divide (a, b):  
 return a / b  
  
calculator = Calculator()  
  
print(calculator.add(10, 5))  
print(calculator.subtract(10, 5))  
print(calculator.multiply(10, 5))  
print(calculator.divide(10, 5))

5)

class player:  
 def \_\_init\_\_(self, name, age, height,weight):  
 self.name = name  
 self.age = age  
 self.height = height  
 self.weight = weight  
 def get\_age(self):  
 return f"{self.name} is age {self.age}"  
 def get\_height(self):  
 return f"{self.name} is height {self.height}"  
 def get\_weight(self):  
 return f"{self.name} is weight {self.weight}"  
p1 = player("David Jones", 25, 175, 75)  
print(p1.get\_age())  
print(p1.get\_height())  
print(p1.get\_weight())

6)

class Employee:  
 def \_\_init\_\_(self, firstname, lastname):  
 self.firstname = firstname  
 self.lastname = lastname  
 def fullname(self):  
 return f"{self.firstname} {self.lastname}"  
 def email(self):  
 return f"{self.firstname.lower()}.{self.lastname.lower()}@company.com"  
emp\_1 = Employee("John", "Smith")  
emp\_2 = Employee("Mary", "Sue")  
emp\_3 = Employee("Antony", "Walker")  
print(emp\_1.fullname())  
print(emp\_2.email())  
print(emp\_3.firstname)

7)

class ones\_threes\_nines:  
 def \_\_init\_\_(self, number):  
 self.ones = number // 1  
 self.threes = number // 3  
 self.nines = number // 9  
n1 = ones\_threes\_nines(5)  
print(n1.nines)  
print(n1.ones)  
print(n1.threes)

8)

class Name:  
 def \_\_init\_\_(self, firstname, lastname):  
 self.firstname = firstname.capitalize()  
 self.lastname = lastname.capitalize()  
  
 def fullname(self):  
 return f"{self.firstname} {self.lastname}"  
  
 def initials(self):  
 return f"{self.firstname[0]}.{self.lastname[0]}"  
  
a1 = Name("john", "SMITH")  
print(a1.firstname)  
print(a1.lastname)  
print(a1.fullname())  
print(a1.initials())

**Classwork\_OOP\_b**

1)

class Book:  
 def \_\_init\_\_(self, title, author):  
 self.title = title  
 self.author = author  
 def get\_title(self):  
 return f"Title: {self.title}"  
 def get\_author(self):  
 return f"Author: {self.author}"  
HP = Book("Harry Potter", "J.K. Rowling")  
print(HP.title)  
print(HP.author)  
print(HP.get\_title())  
print(HP.get\_author())

2)

class User:  
 user\_count = 0  
 def \_\_init\_\_(self, username):  
 self.username = username  
 User.user\_count += 1  
u1 = User("johnsmith10")  
u2 = User("marysue1989")  
u3 = User("milan\_rodrick")  
print(User.user\_count)  
print(u1.username)  
print(u2.username)  
print(u3.username)

3)

class Composer:  
 counter = 0  
 def \_\_init\_\_(self, name, dob, country):  
 self.name = name  
 self.dob = dob  
 self.country = country  
 Composer.counter += 1  
c1 = Composer("Ludvig van Beethoven", 1770, "Germany")  
c2 = Composer("Wolfgang Amadeus Mozart", 1756, "Austria")  
c3 = Composer("Johannes Brahms", 1833, "Germany")  
print(Composer.counter)

4)

class Person:  
 def \_\_init\_\_(self, name, like, hate):  
 self.name = name  
 self.like = like  
 self.hate = hate  
 def taste(self, food):  
 if food in self.like:  
 return f"{self.name} eats the {food} and loves it"  
 elif food in self.hate:  
 return f"{self.name} eats the {food} and hates it"  
 else:  
 return f"{self.name} eats the {food}"  
p1 = Person("Sam", ["ice cream"], ["carrots"])  
print(p1.taste("ice cream"))  
print(p1.taste("cheese"))  
print(p1.taste("carrots"))

5)

class Country:  
 def \_\_init\_\_(self, name, population, area):  
 self.name = name  
 self.population = population  
 self.area = area  
 def is\_big(self):  
 if self.population > 250000000 or self.area > 3000000:  
 return True  
 else:  
 return False  
 def compare\_pd(self, other):  
 if self.population / self.area > other.population / other.area:  
 return f"{self.name} has a larger population density than {other.name}"  
 else:  
 return f"{other.name} has a larger population density than {self.name}"  
australia = Country("Australia", 23545500, 7692024)  
andorra = Country("Andorra", 76098, 468)  
print(australia.is\_big())  
print(andorra.is\_big())  
print(andorra.compare\_pd(australia))

6)

class Programmer:  
 def \_\_init\_\_(self, salary, work\_hours):  
 self.salary = salary  
 self.work\_hours = work\_hours  
 def \_\_del\_\_(self):  
 return f"oof, {self.salary}, {self.work\_hours}"  
 def compare\_prog(prog1, prog2):  
 if prog1.salary < prog2.salary:  
 return f"{prog1.salary} salary is lower"  
 elif prog1.salary > prog2.salary:  
 return f"{prog2.salary} salary is lower"  
 elif prog1.salary == prog2.salary:  
 if prog1.work\_hours > prog2.work\_hours:  
 return f"{prog2.salary} salary is lower"  
 elif prog1.work\_hours < prog2.work\_hours:  
 return f"{prog1.salary} salary is lower"  
prog1 = Programmer(25000, 5)  
prog2 = Programmer(21000, 5)  
print(prog1.\_\_del\_\_())  
print(prog2.salary)  
print(prog1.work\_hours)  
print(Programmer.compare\_prog(prog1, prog2))

7)

class IceCream:  
 def \_\_init\_\_(self, flavor, num\_sprinkles):  
 self.flavor = flavor  
 self.num\_sprinkles = num\_sprinkles  
 self.sweetness\_values = {  
 "Plain": 0,  
 "Vanilla": 5,  
 "ChocolateChip": 5,  
 "Strawberry": 10,  
 "Chocolate": 10  
 }  
 def get\_sweetness(self):  
 return self.sweetness\_values[self.flavor] + self.num\_sprinkles  
 def sweetest\_icecream(icecream\_list):  
 max\_sweeet = 0  
 for icecream in icecream\_list:  
 sweeet = IceCream.get\_sweetness(icecream)  
 max\_sweeet = max(sweeet, max\_sweeet)  
 return max\_sweeet  
ice1 = IceCream("Chocolate", 13)  
ice2 = IceCream("Vanilla", 0)  
ice3 = IceCream("Strawberry", 7)  
ice4 = IceCream("Plain", 18)  
ice5 = IceCream("ChocolateChip", 3)  
print(IceCream.sweetest\_icecream([ice1, ice2, ice3, ice4, ice5]))  
print(IceCream.sweetest\_icecream([ice3, ice1]))  
print(IceCream.sweetest\_icecream([ice3, ice5]))

8)

class Magic:  
 def replace(self, input\_str, old\_char, new\_char):  
 return input\_str.replace(old\_char, new\_char)  
 def str\_length(self, input\_str):  
 return len(input\_str)  
 def trim(self, input\_str):  
 return input\_str.strip()  
 def list\_slice(self, input\_list, index\_range):  
 start, end = index\_range  
 sliced\_list = input\_list[start:end + 1]  
 return sliced\_list if sliced\_list else -1  
magic = Magic()  
print(magic.replace("AzErty-QwErty", "E", "e"))  
print(magic.str\_length("hello world"))  
print(magic.trim(" python is awesome "))  
print(magic.list\_slice([1, 2, 3, 4, 5], (2, 4)))

9)

class OnesThreesNines:  
 def \_\_init\_\_(self, number):  
 self.nines = number // 9  
 self.threes = (number % 9) // 3  
 self.ones = (number % 9) % 3 // 1  
 def \_\_str\_\_(self):  
 return f"nines:{self.nines}, threes:{self.threes}, ones:{self.ones}"  
result1 = OnesThreesNines(10)  
result2 = OnesThreesNines(15)  
result3 = OnesThreesNines(22)  
print(result1)  
print(result2)  
print(result3)

10)

class Employee:  
 def \_\_init\_\_(self, firstname, lastname, salary):  
 self.firstname = firstname  
 self.lastname = lastname  
 self.salary = salary  
  
 @classmethod  
 def from\_string(cls, employee\_string):  
 attributes = employee\_string.split('-')  
 return cls(attributes[0], attributes[1], int(attributes[2]))  
emp1 = Employee("Mary", "Sue", 60000)  
emp2 = Employee.from\_string("John-Smith-55000")  
print(emp1.firstname)  
print(emp1.salary)  
print(emp2.firstname)  
print(emp2.salary)