الجامعة الدولية للعلوم والنهضة

كلية الهندسة المعلوماتية

برمجة نصية (بايثون) – عملي

إعداد:

أ. محمد جراد

أ. خالد الإسماعيل

1. اكتب برنامج فيه صنف يمثل شخص يحوي الخصائص العامة وطريقة لحساب عمره

```
import datetime
class Person:
    def __init__(self, name, birth_year):
        self.name = name
        self.birth_year = birth_year
    def print_data(self):
        print(f"Name: {self.name} and year of birth: {self.birth_year}")
    def calculate_age(self):
        print(f"Age: {datetime.datetime.now().year - self.birth_year}")

p1 = Person("Muhammad Ali", 2000)
p1.print_data()
p1.calculate_age()

p1.name = "Ahmed"
p1.birth_year = 1995
print("\nUpdated Data:")
p1.print_data()
p1.calculate_age()
```

مع التغليف

```python
import datetime
class Person:
    def __init__(self):
        self.__name = None
        self.__birth_year = None
    @property
    def name(self):
        return self.__name
    @name.setter
    def name(self, value):
        if value:  # تحقق بسيط للتأكد من أن الاسم ليس فارغًا
            self.__name = value
        else:
            print("Name cannot be empty.")
    @property
    def birth_year(self):
        return self.__birth_year
    @birth_year.setter
    def birth_year(self, value):
        if value > 0:
            self.__birth_year = value
        else:
            print("Year of birth must be positive.")
    def print_data(self):
        if self.__name is not None and self.__birth_year is not None:
            print(f"Name: {self.__name} and year of birth: {self.__birth_year}")
        else:
            print("Data is incomplete. Please set the name and birth year.")
    def calculate_age(self):
        if self.__birth_year is not None:
            print(f"Age: {datetime.datetime.now().year - self.__birth_year}")
        else:
            print("Birth year is not set.")

p1 = Person()
p1.name = "Muhammad Ali"; p1.birth_year = 2000
p1.print_data(); p1.calculate_age()

p1.name = "Ahmad"; p1.birth_year = 1995
print("\nUpdated Data:"); p1.print_data(); p1.calculate_age()
```

2. كتابة برنامج فيه صنف يمثل عربة التسوق. قم بتضمين طرق لإضافة العناصر وإزالتها وحساب الكمية الإجمالية.

```
class ShoppingCart:
    def __init__(self):
        self.items = {}
    def add_item(self, item_name, quantity):
        if item_name in self.items:
            self.items[item_name] += quantity
        else:
            self.items[item_name] = quantity
    def remove_item(self, item_name):
        if item_name in self.items:
            del self.items[item_name]
    def calculate_total(self):
        return sum(self.items.values())

cart = ShoppingCart()
cart.add_item("Apple", 100)
cart.add_item("Banana", 200)
cart.add_item("Orange", 150)

print("Current Items in Cart:\n", cart.items)
print("Total Quantity:", cart.calculate_total())

cart.remove_item("Orange")
print("\nUpdated Items in Cart after removing Orange:\n", cart.items)
print("Total Quantity:", cart.calculate_total())
```

3. اكتب برنامج فيه صنف يمثل دائرة وصنف اخر أسطوانة يرث منه وكتابة الطرق الخاصة بمحيط ومساحة كل منهما

```python
import math
class Circle:
    def __init__(self, radius):
        self.radius = radius
    def perimiter(self):
        return 2 * math.pi * self.radius
    def area(self):
        return math.pi * (self.radius ** 2)

class Cylinder(Circle):
    def __init__(self, radius, height):
        super().__init__(radius)
        self.height = height
    def lateral_area(self):
        return 2 * math.pi * self.radius * self.height
    def total_area(self):
        return 2 * self.area() + self.lateral_area()
    def volume(self):
        return self.area() * self.height
circle = Circle(5)
print(f"Perimiter: {circle.perimiter():.2f}")
print(f"Area: {circle.area():.2f}")

cylinder = Cylinder(5, 10)
print(f"Total Area: {cylinder.total_area():.2f}")
print(f"Volume: {cylinder.volume():.2f}")
```

4. اكتب برنامج فيه صنف يمثل شخص وصنف اخر طالب يرث منه فيه طريقة لحساب المعدل ،وصنف اخر يمثل موظف يرث من صنف الشخص أيضاً، وفيه طريقة لحساب الأجر الشهري مع المكافآت.

```
class Person:
    def __init__(self, name):
        self.name = name
    def __str__(self):
        return f"{self.name}"

class Student(Person):
    def __init__(self, name, grades):
        super().__init__(name)
        self.grades = grades
    def calculate_average(self):
        return sum(self.grades) / len(self.grades)
    def __str__(self):
        return f"{super().__str__()} , Average Grade:
{self.calculate_average()}"

class Employee(Person):
    def __init__(self, name, base_salary):
        super().__init__(name)
        self.base_salary = base_salary
        self.reward = 0
    def set_reward(self, reward):# تعيين قيمة المكافأة
        self.reward = reward
    def calculate_salary(self):
        return self.base_salary + self.reward
    def __str__(self):# magic method
        return f"{super().__str__()} , Salary: {self.calculate_salary()}$"

student = Student("Muhammad", [85, 90, 78, 92])
print("Student Info:")
print(student)

employee = Employee("Khaled", 100)
employee.set_reward(1200)
print("\nEmployee Info:")
print(employee) # __str__ يتم تنفيذ دالة
```

5.اكتب برنامج فيه صنف يمثل مديريرث من صنفين آخرين (صنف شخص وصنف موظف)، قم بتضمين في كل صنف أب طريقة ما ، ليتم توريثها الى الصنف الابن. (مفهوم الوراثة المتعددة)

```
class Person:
    def __init__(self, name, birth_year):
        self.name = name
        self.birth_year = birth_year
    def calculate_age(self):
        age = 2024 - self.birth_year
        print(f"Age: {age}")

class Employee:
    def __init__(self, job_title, salary):
        self.job_title = job_title
        self.salary = salary
    def calculate_annual_salary(self):
        annual_salary = self.salary * 12
        print(f"Annual Salary: ${annual_salary}")

class Manager(Person, Employee):
    def __init__(self, name, birth_year, job_title, salary, department):
        super().__init__(name, birth_year)     # استدعاء باني الصنف الأول
        Employee.__init__(self, job_title, salary) # استدعاء باني الصنف الثاني
        self.department = department

manager = Manager("Muhammad", 2000, "Project Manager", 8000, "IT")
manager.calculate_age()
manager.calculate_annual_salary()
```

6.اكتب برنامج فيه صنف يمثل كتاب وصنف اخر يمثل مؤلف الكتاب (مفهوم التركيب)

```
class Author:
    def __init__(self, name, birth_year):
        self.name = name
        self.birth_year = birth_year
    def __str__(self):
        return f"Author Name: {self.name},Author Born: {self.birth_year}"

class Book:
    def __init__(self, title, author, publication_year):
        self.title = title
        self.author = author
        self.publication_year = publication_year
    def __str__(self):
        return f"Book: '{self.title}', {self.author}, Published:
{self.publication_year}"

author = Author("Muhammad Ali", 1980)
book = Book("Book1", author, 2020)
print(book)
```

7. اكتب برنامجًا لإدارة بنك يحتوي على الطرق التالية:

- انشاء حساب جديد
- طريقة لإيداع مبلغ ما وطريقة لسحب مبلغ
- طريقة لمعرفة الرصيد
- طريقة لعرض كل الحسابات
- طريقة لإغلاق حساب ما

```
class Bank:
    def __init__(self):
        self.customers = {}
    def create_account(self, account_number, initial_balance=0):
        if account_number in self.customers:
            print("Account number already exists.")
        else:
            self.customers[account_number] = initial_balance
            print("Account created successfully.")
    def deposit(self, account_number, amount):
        if account_number in self.customers:
            if amount > 0:
                self.customers[account_number] += amount
                print("Deposit successful.")
            else:
                print("Deposit amount must be positive.")
        else:
            print("Account number does not exist.")
    def make_withdrawal(self, account_number, amount):
        if account_number in self.customers:
            if amount > 0:
                if self.customers[account_number] >= amount:
                    self.customers[account_number] -= amount
                    print("Withdrawal successful.")
                else:
                    print("Insufficient funds.")
            else:
                print("Withdrawal amount must be positive.")
        else:
            print("Account number does not exist.")
```

```
def check_balance(self, account_number):
        if account_number in self.customers:
            balance = self.customers[account_number]
            print(f"Account balance: ${balance:.3f}")
        else:
            print("Account number does not exist.")

    def list_all_accounts(self):
        if self.customers:
            print("All accounts and their balances:")
            for account_number, balance in self.customers.items():
                print(f"Account Number: {account_number}, Balance: ${balance:.2f}")
        else:
            print("No accounts available.")

    def close_account(self, account_number):
        if account_number in self.customers:
            del self.customers[account_number]
            print("Account closed successfully.")
        else:
            print("Account number does not exist.")

bank = Bank()
bank.create_account("12345", 500)
bank.create_account("67890", 300)

bank.deposit("12345", 200)
bank.make_withdrawal("67890", 100)
bank.check_balance("12345")
bank.check_balance("67890")

bank.list_all_accounts()

bank.close_account("67890")
bank.list_all_accounts()
```