Northumbria University NEWCASTLE

Department of Computer & Information Sciences

ASSESSMENT 1 BRIEF	
Module Title:	Advanced Programming
Module Code:	KV5031
Academic Year / Semester:	2025/26 Semester 1
Module Tutor / Email (all queries):	Nick Dalton nick.dalton@northumbria.ac.uk
% Weighting (to overall module):	50%
Assessment Title:	Object-oriented code exercises
Date of Handout to Students:	Seen blackboard
Mechanism for Handout:	Module Blackboard Site & Seminar in Week 3
Deadline for Attempt Submission by Students:	10/Nov/2025 13:00
Mechanism for Submission:	Document upload to Module Blackboard Site
Submission Format / Word Count	Please upload your solution code to the form on the module blackboard/ word count N/A.
Date by which Work, Feedback and Marks will be returned:	20 working days
Mechanism for return of Feedback and Marks:	Mark and individual written feedback sheet will be uploaded to the Module Site on Blackboard. For further queries please email module tutor.

LEARNING OUTCOMES

The learning outcomes (LOs) for this module are:-

LO1 Knowledge & Understanding: Demonstrate an understanding of the basic principles of programming using an appropriate programming language, including the use of variables, conditions, loops, subprograms

LO2 Demonstrate an understanding of abstraction mechanisms and structured data types

LO3 Intellectual / Professional skills & abilities: Create reliable and maintainable software using appropriate code standards.

LO4 Personal Values Attributes (Global / Cultural awareness, Ethics, Curiosity) (PVA): Solve problems using a structured approach.

This assessment addresses learning outcomes LO1, LO2, LO3 and LO4.

Nature of the submission required:

See the section how to submit work.

Instructions to students:

This is an individual piece of work.

Referencing Style:

N/A

Expected size of the submission:

TBD

Academic Conduct:

You must adhere to the university regulations on academic conduct. Formal inquiry proceedings will be instigated if there is any suspicion of misconduct or plagiarism in your work. Refer to the University's regulations on assessment if you are unclear as to the meaning of these terms. The latest copy is available on the university website.

Northumbria University NEWCASTLE

Start here

Auto-marking code

This assessment makes use of auto-marking code in the form of a unit test. This will mark the code you provide and give you feedback in real time. The unit test contains elements which allow us to check you are not cheating or using AI to complete your assessments. **Do not interfere with this code**.

Each question should be in the appropriate Python file. Cut and paste the code into the boxes on the assessment form in blackboard. Do NOT give a link to your code or upload the code in the document.

Use of AI and plagiarism spot checks

Collusion and the use of AI for assessments is strictly forbidden. To combat AI usage and plagiarism, at the end of the assessment**, a list of students will be randomly chosen** to attend a follow-up workshop. In the workshop, the selected students will be expected to explain or reproduce one of their submitted answers to a tutor.

Creating a class is a common, everyday activity for a developer. An inability to create a class on demand, or claiming "I just forgot how I did that; it was so long ago when I completed the assessment," will not be acceptable. Spot-checked **students who are unable to reproduce their submitted work will receive a score of zero** for the assessment at the discretion of the tutor present.

Students who fail to attend their workshop after being given two opportunities to do so will receive a score of zero marks. If you have completed the questions below, you should find this check a trivial activity. There is nothing to be worried about.

Extensions

Students **without** a SAP, who award themselves automatic no question extensions will be added to the spot-check list.

We Are Here to Help

Remember, we are here to support you throughout your learning journey. If you have any questions or concerns about any aspect of the course, **don't hesitate to talk to a tutor during the workshops**. Whether you need clarification on a specific topic, help with an assignment, or just someone to explain things more clearly, feel free to ask.

You can approach any tutor in any workshop—nothing is too trivial or "stupid" to ask about. We understand that object orientation can be particularly challenging, especially if you're

new to programming. Our goal is to help you understand and succeed, so please take advantage of the resources and assistance available to you.

QUESTION

1. You are required to create a Python class named Student that models a student. The class should have the following attributes and methods:
    a. **Attributes:**
        i. **name** : A string representing the student's name.
        ii. **student_id**: An integer representing the student's unique identification number.
        iii. **courses**: A list of strings representing the courses the student is enrolled in.
        iv. **grades**: A dictionary where the keys are course names and the values are the grades the student has received in those courses.
    b. **Methods:**
        i. **__init__(self, name: str, student_id: int)**: Initializes a new Student object with the provided name and student ID. The courses list should be empty, and the grades dictionary should be empty when a student object is created.
        ii. **enroll(self, course_name: str)**: Enrolls the student in a new course by adding the course name to the courses list. If the student is already enrolled in the course, the method should do nothing.
        iii. **add_grade(self, course_name: str, grade: float)**: Adds a grade for a specific course to the grades dictionary. If the student is not enrolled in the course, the method should raise a ValueError with the message "Student is not enrolled in the course. ".
        iv. **get_gpa(self)**: Calculates and returns the student's GPA as the average of all grades in the grades dictionary. If the student has no grades, the method should return None.
        v. **str(self)**: Returns a string representation of the Student object in the format: "Student Name: <name>, ID: <student_id>, Courses: <courses>".

```
# Creating a new student

student = Student("Alice", 1234)

# Enrolling the student in courses
```

```
student.enroll("Math")

student.enroll("History")

# Adding grades

student.add_grade("Math", 90)

student.add_grade("History", 85)

# Getting the GPA

print(student.get_gpa())  # Output: 87.5

# Displaying the student's information

print(student)  # Output: "Student Name: Alice, ID: 1234,
Courses: ['Math', 'History']"
```

QUESTION

2. You are required to create two Python classes: Patient and DayPatient in a file called HospitalManagement.py. The DayPatient class should inherit from the Patient class and override one of its methods:
    a. Class **Patient**
        i. Attribute
            1. **name**: A string representing the patient's name.
            2. **age**: An integer representing the patient's age.
            3. **patient_id**: An integer representing the patient's unique identification number.
            4. **illness**: A string representing the illness the patient is being treated for.
            5. **admission_date**: A string representing the date the patient was admitted.
        ii. **Methods**
            1. **__init__(self, name: str, age: int, patient_id: int, illness: str, admission_date: str)**: Initializes a new Patient object with the provided name, age, patient ID, illness, and admission date.
            2. **get_details(self)**: Returns a string containing the patient's name, age, patient ID, illness, and admission date in the format:
            3. "Patient Name: <name>, Age: <age>, ID: <patient_id>, Illness: <illness>, Admission Date: <admission_date>".
            4. **get_treatment_plan (self)**: Returns a string representing a generic treatment plan for the patient. This method should return:

5. "The treatment plan for this patient will be provided after further assessment.".Attribute

b. Class **DayPatient**

i. Attribute

1. **name**: A string representing the patient's name.
2. **age**: An integer representing the patient's age.
3. **patient_id**: An integer representing the patient's unique identification number.
4. **illness**: A string representing the illnes the patient is being treated for.
5. **admission_date**: A string representing the date the patient was admitted.
6. **discharge_time**: A string representing the expected discharge time for the day patient.

ii. **Methods**

1. **__init__(self, name: str, age: int, patient_id: int, illness: str, admission_date: str, discharge_time: str)**: Initializes a new DayPatient object using the constructor of the Patient class and additionally initializes the discharge_time attribute.
2. **get_details(self)**: This method should override the get_details method from the Patient class. It should return the details of the day patient, including the discharge time, in the format:
3. "Day Patient Name: <name>, Age: <age>, ID: <patient_id>, Illness: <illness>, Admission Date: <admission_date>, Discharge Time: <discharge_time>".
4. **get_treatment_plan(self)**: This method should override the get_treatment_plan method from the Patient class. It should return:
5. "The treatment plan includes day procedures and the patient will be discharged by <discharge_time>.".

QUESTION

You are required to create a Python class named BankAccount that models a simple bank account in a file called *test_bank_account.py* . The class should **be fully encapsulated**:

a. **Class: BankAccount**

b. The BankAccount class should have the following private attributes and public methods:

i. **account_number** : A string representing the unique account number.

ii. **account_holder**: A string representing the name of the account holder.

iii. **balance** : A float representing the account balance.

iv. **account_type**: A string representing the type of account (e.g., “Savings”, “Checking”).

v. **__init__(self, account_number: str, account_holder: str, balance: float, account_type: str)**: Initializes a new BankAccount object with the provided account number, account holder name, initial balance, and account type.

QUESTION

You are required to create a Python class named Book that models a book in a library system. The class should be implemented as a data class using the @dataclass decorator, which provides automatic generation of common methods such as __init__, __repr__, and __eq__.

a. **Class: Book**

- The Book data class should have the following attributes:

b. **Attributes:**

- **title**: A string representing the title of the book.
- **author** : A string representing the name of the author .
- **isbn**: A string representing the book’s ISBN (International Standard Book Number).
- **publication_year** : An integer representing the year the book was published.
- **copies_available**: An integer representing the number of copies of the book that are currently available in the library.
- **Plus one more of your own devising**,

c. **Additional Requirements:**

- **Adding Type Annotations:** Ensure that all attributes are properly type-annotated.

2. **Creating Methods:**

- Implement a method named is_available(self) -> bool that returns True if there is at least one copy of the book available, otherwise False.
- Implement a method named borrow_book(self) that decreases the copies_available by 1 if the book is available. If no copies are available, raise a ValueError with the message “No copies available to borrow.”
- Implement a method named return_book(self) that increases the copies_available by 1.

QUESTION

You are required to create a Python class named Vector2D that models a two-dimensional vector. The class should override the infix operators to enable vector addition, subtraction, and scalar multiplication.

1. Class: Vector2D
    a. The Vector2D class should have the following attributes and methods:
2. Attributes:
    a. x: A float representing the x-coordinate of the vector.
    b. y: A float representing the y-coordinate of the vector.

The class vector should be able to operate like this

```
# Creating two vectors
vector1 = Vector2D(2, 3)
vector2 = Vector2D(4, 1)

# Adding two vectors
vector_sum = vector1 + vector2
print(vector_sum)  # Output: Vector2D(x=6, y=4)

# Subtracting two vectors
vector_diff = vector1 - vector2
print(vector_diff)  # Output: Vector2D(x=-2, y=2)

# Scalar multiplication
vector_scaled = vector1 * 3
print(vector_scaled)  # Output: Vector2D(x=6, y=9)
```

Requirements:

- Override the + operator to perform vector addition .
- Override the – operator to perform vector subtraction.
- Override the * operator to perform scalar multiplication.
- Provide a meaningful string representation of the vector using the __str__ method.

Submission:

Submit your Python code implementing the Vector2D class with the above attributes and methods. Ensure that the infix operators are properly overridden and that your implementation handles various operations correctly.

QUESTION

You are required to create a Python decorator named time_logger that measures and logs the execution time of a function. The decorator should be applicable to any function, and it should print out the time taken for the function to execute.

Requirements:

1. **Decorator Name**: time_logger
2. **Functionality**:
    a. The decorator should measure the time taken for a function to execute.
    b. After the function has executed, the decorator should print a message in the format:
    c. "Function '<function_name>' executed in <time_taken> seconds. "
    d. The <function_name> should be the name of the decorated function, and <time_taken> should be the execution time rounded to 4 decimal places.
3. **The decorator should work with functions that take any number of positional and keyword arguments.**

- You may use the time module for measuring execution time.
- Ensure that the decorator does not alter the return value of the decorated function.
- The decorator should handle any function signature (i.e., functions with different numbers and types of arguments).

Example usage

```
import time

@time_logger
def slow_function(seconds):
    time.sleep(seconds)
    return "Function complete!"

@time_logger
def add(a, b):
    return a + b

# Calling the decorated functions
result1 = slow_function(2)
# Output: Function 'slow_function' executed in 2.000x seconds. (Where x is
                the difference due to processing time)

result2 = add(5, 7)
```

```
# Output: Function 'add' executed in 0.000x seconds. (Where x is the
                  difference due to processing time)
```

TO REDUCE STRESS

The following questions do not have an automatic mark. If you have made it this far and answer all questions then you have passed assessment 1 (of 2) with at least 40% The following questions exist to improve your mark and allow students to get higher grades (with more levels of difficulty).

QUESTION

lass choice Three different AI have produced the following skeleton classes.

Choice A Student-centric model

```
from dataclasses import dataclass, field

from typing import Dict, Set



@dataclass(frozen=True)

class Module:

    code: str

    title: str



@dataclass

class Student:

    sid: str

    name: str

    modules: Set[str] = field(default_factory=set)  # store module codes



    def enrol(self, module: Module) -> None:
```

```
        self.modules.add(module.code)


class Registrar:
    """Holds canonical records and runs queries that cut across Students."""
    def __init__(self) -> None:
        self.students: Dict[str, Student] = {}
        self.modules: Dict[str, Module] = {}

    def add_student(self, s: Student) -> None:
        self.students[s.sid] = s

    def add_module(self, m: Module) -> None:
        self.modules[m.code] = m

    def enrol(self, sid: str, module_code: str) -> None:
        self.students[sid].enrol(self.modules[module_code])

    def students_in_module(self, module_code: str) -> list[Student]:
        # Must scan all students because ownership is student-side
        return [s for s in self.students.values() if module_code in s.modules]

# --- Example usage
reg = Registrar()
```

```
m1 = Module("CSC101", "Intro to CS"); reg.add_module(m1)

s1 = Student("S001", "Aisha"); s2 = Student("S002", "Ben")

reg.add_student(s1); reg.add_student(s2)

reg.enrol("S001", "CSC101")

print([s.name for s in reg.students_in_module("CSC101")])  # ['Aisha']
```

Choice B MODULE CENTERED

```
from dataclasses import dataclass, field

from typing import Dict, Set


@dataclass(frozen=True)

class Student:

    sid: str

    name: str


@dataclass

class Module:

    code: str

    title: str

    student_ids: Set[str] = field(default_factory=set)
```

```
    def enrol(self, sid: str) -> None:
        self.student_ids.add(sid)


class Registry:
    """Modules own rosters, so module-centric queries are cheap."""
    def __init__(self) -> None:
        self.students: Dict[str, Student] = {}
        self.modules: Dict[str, Module] = {}

    def add_student(self, s: Student) -> None:
        self.students[s.sid] = s

    def add_module(self, m: Module) -> None:
        self.modules[m.code] = m

    def enrol(self, sid: str, module_code: str) -> None:
        self.modules[module_code].enrol(sid)

    def students_in_module(self, module_code: str) -> list[Student]:
        mod = self.modules[module_code]
        return [self.students[sid] for sid in mod.student_ids]
```

```
# --- Example usage

reg = Registry()

reg.add_student(Student("S001", "Aisha"))

reg.add_student(Student("S002", "Ben"))

reg.add_module(Module("CSC101", "Intro to CS"))

reg.enrol("S001", "CSC101")

print([s.name for s in reg.students_in_module("CSC101")])  # ['Aisha']
```

Choice C: Course-centric model (course orchestrates everything)

```
from dataclasses import dataclass, field

from typing import Dict, Set



@dataclass(frozen=True)

class Lecturer:

    lid: str

    name: str



@dataclass(frozen=True)

class Student:

    sid: str

    name: str



@dataclass(frozen=True)
```

```
class Module:
    code: str
    title: str


class Course:
    """Course is the single source of truth for relationships."""
    def __init__(self, cid: str, title: str) -> None:
        self.cid = cid
        self.title = title
        self.modules: Dict[str, Module] = {}
        self.students: Dict[str, Student] = {}
        self.lecturers: Dict[str, Lecturer] = {}
        # central enrolment mapping: module_code -> set of student_ids
        self._roster: Dict[str, Set[str]] = {}

    # --- structure management
    def add_module(self, m: Module) -> None:
        self.modules[m.code] = m
        self._roster.setdefault(m.code, set())

    def add_student(self, s: Student) -> None:
        self.students[s.sid] = s
```

```
    def add_lecturer(self, l: Lecturer) -> None:
        self.lecturers[l.lid] = l

    # --- operations
    def enrol(self, sid: str, module_code: str) -> None:
        if sid not in self.students:
            raise KeyError(f"Unknown student: {sid}")
        if module_code not in self.modules:
            raise KeyError(f"Unknown module: {module_code}")
        self._roster[module_code].add(sid)

    def students_in_module(self, module_code: str) -> list[Student]:
        return [self.students[sid] for sid in self._roster.get(module_code, set())]

    def modules_for_student(self, sid: str) -> list[Module]:
        codes = [code for code, sids in self._roster.items() if sid in sids]
        return [self.modules[c] for c in codes]

# --- Example usage
course = Course("CS-BSc", "BSc Computer Science")
course.add_module(Module("CSC101", "Intro to CS"))
course.add_module(Module("CSC205", "Data Structures"))
course.add_student(Student("S001", "Aisha"))
```

```
course.add_student(Student("S002", "Ben"))

course.enrol("S001", "CSC101")

course.enrol("S001", "CSC205")

print([s.name for s in course.students_in_module("CSC101")])  # ['Aisha']

print([m.code for m in course.modules_for_student("S001")])   # ['CSC101', 'CSC205']
```

In your opinion which way of the above is a better way to set up the application ?

Write a short paragraph to explain to your boss and team why your chosen way is best. To avoid AI please write in a way that your answer is readable in the 7th grade. You can check this via https://hemingwayapp.com . That is your text should be pasteable into the editor and there should be no highlighted lines. You should be writing from the view point of computer science object orientated programming. Questions you can ask yourself. Which is a better example of OOP, which is more usable.

Reflect on the alternative designs presented above. In your opinion, which approach provides a better way to structure the application?
Write a short paragraph, as if addressing your manager and development team, explaining why your chosen design is preferable. That is, your text should be pasteable into the Hemingway Editor https://hemingwayapp.com and there should be no highlighted lines. Marks will be deducted if your submission contains highlighted items.
When preparing your answer, write from the perspective of computer science and object-oriented programming. Consider questions such as: Which design demonstrates stronger use of OOP principles? Which approach is more usable, maintainable, or scalable?

Question

For your chosen skeleton

find something to subclass and write the code for that subclass with a method the parent and child class both override

Question

Examine the specification for Assessment 2: Here you will find the specification for a **On-Licence Housing Allocation System**. Read the specification and draw the wireframes for the app GUI you propose to implement the desktop application. The second assessment is programming this app using your knowledge of Pyside6. The final result will be marked on how close to the app your final result is – deviations are expected and normal.

The wireframe is normally delivered as a Powerpoint. If you used another app please supply the PDF and the source files.

This is not a measure of your ability to draw but your ability to plan and envision. One of the hardest things was wondering what to make. This short exercise is intended to help you be ready for the next assessment. It also makes using an AI generator that much more difficult and painful.

END OF QUESTIONS

Assessment Regulations

You are advised to read the guidance for students regarding assessment policies. They are available online here. (http://www.northumbria.ac.uk/about-us/university-services/academic-registry/quality-and-teaching-excellence/assessment/guidance-for-students/)

Late submission of work

Where coursework is submitted without approval, after the published hand-in deadline, the following penalties will apply.

- For coursework submitted up to 1 working day (24 hours) after the published hand-in deadline without approval, **10% of the total marks available for the assessment** (i.e.,100%) **shall be deducted** from the assessment mark.
- Coursework submitted more than 1 working day (24 hours) after the published hand-in deadline without approval will be regarded as not having been completed. **A mark of zero will be awarded for the assessment and the module will be failed**, irrespective of the overall module mark.

The full policy can be found here.

Academic misconduct

In all assessed work you should take care to ensure that the work you submit is your own. The University takes academic dishonesty and cheating very seriously, and it is your responsibility to ensure that you don't attempt to cheat or become victim to cheating.

There are many different forms of academic misconduct or 'cheating'. Plagiarism is the most common and both the University library and your academic tutors are able to provide further guidance on proper citation and referencing in your assessed work.

- The full Academic Misconduct Policy is available here.
- Useful guidance for avoiding academic misconduct can be found here.

Remember, this is an **INDIVIDUAL** assessment and should be entirely your own work. Where you have used someone else's words (quotations), they should be correctly quoted or referenced. Help regarding referencing and guidance on avoiding plagiarism can be found in the Study Skills section of the module site on Blackboard.

Opportunities for Formative and Final Feedback

During classes, formative discussions will provide guidance and support for all aspects relating to the module assessment and successfully evidencing the learning outcomes. Unmoderated marks and summative feedback are expected to be returned within 20 working days after the final submission deadline (see cover sheet for details). Please use the **WORKSHOPS** as the primary means of clarification, guidance and support on any aspect relating to this assessment.