THE UNIVERSITY OF SYDNEY

Room Number	________
Seat Number	________
Student Number	|__|__|__|__|__|__|__|__|__|

ANONYMOUSLY MARKED

(Please do not write your name on this exam paper)

MOCK EXAM PAPER

(This is not a real exam and not worth any marks)

This is practice to see the layout and example questions for your final exam

MOCK EXAMINATION

Semester 2 - Mock, 2025

INFO1110 Introduction to Programming

EXAM WRITING TIME: 2 hours

READING TIME: 10 minutes

EXAM CONDITIONS:
This is a CLOSED book exam - no material permitted

MATERIALS PERMITTED IN THE EXAM VENUE:
(No electronic aids are permitted e.g. laptops, phones)
None

MATERIALS TO BE SUPPLIED TO STUDENTS:
Answer sheet: Gradescope MCQ (single-sided - 100 Qs)

INSTRUCTIONS TO STUDENTS:

Please read these instructions very carefully. This mock exam has two sections.
- Section A: Foundation knowledge (20 MCQ, 2 code-writing Questions)
- Section B: Advanced knowledge (6 code-writing questions)

You only need to **complete one section**, as follows:

- If you have **not yet achieved** an OK result or better in the Foundation Tests held during the semester: You must attempt section A. This will be used to determine if you have achieved an OK result in Foundation knowledge, and so whether you have met the criteria required for a Passing grade..
- If you complete section A, and are confident in the outcome, then you can choose to also attempt some or all of section B, though this will only be marked if you achieve an OK result or better in section A, so you should first focus on ensuring that section A is completed carefully.
- If you have **already achieved** an OK result or better in the Foundation Tests held during the semester: then you do **NOT** need to attempt the questions in section A (i.e. leave this section blank), and ***you should only answer the questions in Section B***.

Please tick the box to confirm that your examination paper is complete. ☐

For Examiner Use Only

Q	Mark
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	

Total ________

Section A: Foundation Knowledge begins here. There are 22 total Questions.

Only complete this section if you <u>do not</u> have OK or above in the Foundation Test

The multiple-choice questions (Q1-Q20) are worth 1 mark each

The coding questions (Q21-Q22) are worth 3 marks each

(1) What will the following code print?

```
a = 5
b = 10
print("The sum of {} and {} is {}.".format(a, b, a+b))
```

(a) `The sum of a and b is a + b.`	[✓ (b) `The sum of 5 and 10 is 15.`]
(c) `The sum of 5 and 10 is 5 + 10.`	(d) `The sum of a and b is 15.`

(2) What will the following code output?

```
a = not (False or False) and (True or False)
b = True and (not False and (True or False))
c = (True or (False and True)) and (not False or True)
print(a, b, c)
```

(a) `False True False`	(b) `True False False`
(c) `False True True`	[✓ (d) `True True True`]

(3) What will the following code output if the user enters 5 as the first input and 3 as the second input to the program? Note that the user's input to the program has been indicated by bolded text.

```
num_1 = input('Enter first number: ')
num_2 = input('Enter second number: ')
print(num_1, '+', num_2, '+ 10 =', int(num_1 + num_2) + 10)
```

(a) `Enter first number:` **5** `Enter second number:` **3** `5 + 3 + 10 = 5310`	(b) `Enter first number:` **5** `Enter second number:` **3** `5 + 3 + 10 = 810`
[✓ (c) `Enter first number:` **5** `Enter second number:` **3** `5 + 3 + 10 = 63`]	(d) `Enter first number:` **5** `Enter second number:` **3** `5 + 3 + 10 = 18`

(4) Consider the code fragment below. It has a missing piece of code indicated by XXX.

Which of the following options complete the expression to correctly determine whether or not number is even?

```
number = 14
is_even = XXX
print(is_even)
print(f"The number is even: {is_even}")
```

[✓ (a) `number % 2 == 0`]	(b) `number % 2 == 2`
(c) `number // 2 == 2`	(d) `number // 2 == 0`

(5) Consider the code fragment below. It has a missing piece of code indicated by XXX. Complete the print statement to output:

```
The score is 10 out of 20
```

```
score = 10
total = 20
print( XXX )
```

[✓ (a) `"The score is", score, "out of", total`]
(b) `"The score is" + str(score) + "out of" + str(total)`
(c) `"The score is ", score, " out of ", + total`
(d) `"The score is " + score, "out of", str(total)`

(6) What will be the output of the program after the following code fragment is executed?

```
numbers = [3, 1, 2]
sum = 0

for i in range(3):
    sum = sum + numbers[i]
    numbers[i] = sum

print(numbers)
```

(a) `[3, 1, 2]`	(b) `[6, 5, 8]`	[✓ (c) `[3, 4, 6]`]	(d) `[3, 2, 4]`

(7) What will be the output of the program after the following code fragment is executed?

```
cities = ['Sydney', 'Melbourne', 'Perth']
indices = [1, 2, 0]

for i in indices:
    print(cities[i])
```

(a) `Sydney`
`Melbourne`
`Perth`
	(b) `Melbourne`
	`Sydney`
	`Perth`
		[✓ (c)] `Melbourne`
		`Perth`
		`Sydney`
			(d) `Perth`
			`Sydney`
			`Melbourne`

(8) If the missing lines are filled in correctly, the program should put the number of times the letter 'a' appears in each word in all_words into the list frequencies. For example, if all_words was ['apple', 'attack', 'aaaah'], frequencies would be [1, 2, 4] after the execution of the program.

```
1        all_words = ['apple', 'attack', 'aaaah']
2        frequencies = [0, 0, 0]
3
4        # ( code MAY go here )
5        for # ( code WILL go here ) :
6            # ( code WILL go inside FOR )
7
8        print(frequencies)
```

Which of the following options is the correct missing code? **Note**: string.count('a') returns the total number of occurrences of 'a' in string.

(a)	`4  # no code`
	`5  for word in all_words:`
	`6      frequencies = word.count('a')`

(b)	`4  i = len(all_words)`
	`5  for word in all_words:`
	`6      frequencies[i] = word.count('a')`

(c)	`4  # no code`
	`5  for i in range(len(all_words)):`
	`6      frequencies[i] = all_words[i].count('a')`

(d)	`4  # no code`
	`5  for i in frequencies:`
	`6      frequencies[i] = all_words[i].count('a')`

(9) What will be the output of the program after the following code fragment is executed?

```
i = 10
while i > 0:
    i = i // 2
    if i == 5:
        continue
    print(i, end=' ')
```

(a) 10 5 2 1	(b) 10 5 2 1	(c) 10 5 2 1	[✓ (d)] 2 1 0

(10) If the missing lines are correctly filled in, the program should continuously take in **words** from the user and append them to a list `valid_words` until the user enters a word that has **more than 5 characters.** Words longer than 5 characters should <u>**NOT**</u> be included in `valid_words`. Below is an example output of the program, where **bolded** text is input to the program:

```
Enter a word: cat
Enter a word: dog
Enter a word: elephant
Valid Words: ['cat', 'dog']
```

```
1         valid_words = []
2         keep_running = True
3
4         while keep_running:
5             word = input('Enter a word: ')
6             if len(word) # ( code WILL go here ) 5:
7                 # ( code WILL go inside IF )
8             # ( code MAY go here )
9             # …
10
11        print('Valid words:', valid_words)
```

Which of the following options is the correct missing code? **(Filled code is bolded.)**

(a)
```
6     if len(word) <= 5:
7         keep_running = True
8     else:
9         valid_words.append(word)
```

[✓ (b)]
```
6     if len(word) > 5:
7         keep_running = False
8     else:
9         valid_words.append(word)
```

(c)
```
6     if len(word) > 5:
7         keep_running = False
8     valid_words.append(word)
```

(d)
```
6     if len(code) <= 5:
7         keep_running = False
8     else:
9         valid_words.append(word)
```

(11) What will be the output of the program after the following code fragment is executed?

```
def set_name():
    name = 'Alice'
    print(name)

def print_name(other_name: str):
    print(other_name)

name = 'Bob'
set_name()
print_name(name)
print(name)
```

(a) Bob Bob Bob	(b) Alice Alice Alice	(c) Alice Alice Bob	[✓ (d)] Alice Bob Bob

(12) What will be the output of the program after the following code fragment is executed?

```
def function_1(num_1: int):
    x = num_1 * 2
    return num_1 + 2

def function_2(num_2: int):
    y = num_2 // 2
    return num_2 + 1

def main():
     x = 3
     y = 4
     print(function_1(x))
     print(function_2(y))
     print(x, y)

main()
```

[✓ (a)] 5	(b) 8	(c) 5	(d) 5
5	3	5	5
3 4	3 4	5 5	6 2

(13) What will be the output of the program after the following code fragment is executed?

```
MULTIPLIER = 3

def calculate(x: int, y: int):
    if y <= 2:
        y = MULTIPLIER
    result = x * y
    return result

def main():
     result = calculate(4, 3)
     print(calculate(3, 2))
     print(result)
main()
```

(a) 12	(b) 6	[✓ (c)] 9	(d) 9
9	12	12	9
12			

(14) What will be the output of the program after the following code fragment is executed?

```
def check_code(code: int):
    if code <= 0:
         return 'Impostor!'
    elif code > 100:
         return 'Valid code.'
    else:
         return 'You are a VIP!'

 print(check_code(1234))
 print(check_code(100))
 print(check_code(-10))
```

(a) Impostor!	(b) You are a VIP!	(c) Valid code.	[✓ (d)] Valid code.
Valid code.	Valid code.	Valid code.	You are a VIP!
You are a VIP!	Impostor!	Impostor!	Impostor!

(15) If the missing lines are correctly filled in, the program should define 2 functions, named `enter_level` and `finish_level`.

- `enter_level` should display the user's score when starting the level.
- `finish_level` should display the user's score when finishing the level. Then, it should display how much score they gained from finishing the level.

```
Level 3 started: Score at 100.
Level 3 finished: Score at 400.
You gained 300 points!
```

```
1    def # ( code WILL go here ) :
2        print(f'Level {level} started: Score at {start}.')
3
4    def # ( code WILL go here ) :
5        difference = end - start
6        print(f'Level {level} finished: Score at {end}.')
7        print(f'You gained {difference} points!')
8
9    def main():
10       level_number = 3
11       start_score = 100
12       enter_level(level_number, start_score)
13
14       # Time passes as player plays level...
15       end_score = 400
16       finish_level(level_number, start_score, end_score)
17
18   main()
```

Which of the following options is the correct missing code?

(a)	`1   def enter_level(level: int, start: int, end: int):`
	`...`
	`4   def  finish_level(level: int, start: int, end: int):`

(b)	`1   def enter_level(level: int, start: int):`
	`...`
	`4   def  finish_level(level: int, end: int, start: int):`

[✓ (c)]	`1   def enter_level(level: int, start: int):`
	`...`
	`4   def  finish_level(level: int, start: int, end: int):`

(d)	`1   def enter_level(level: int, start: int, end: int):`
	`...`
	`4   def  finish_level(level: int, start: int):`

(16) Look at the four function implementations defined below. The purpose of the function is to count the number of strings in a list. Which implementation will cause an error?

```
(a) def count_strings (items):
        count = 0
        for i in items:
            if type(i) == str:
                count += 1
        return count
```

```
(b)def count_strings (items):
        count = 0
        for i in range(len(items))
            if type(items[i]) == int:
                count += 0
            else:
                count += 1
        return count
```

```
(c) def count_strings (items):
        i = len(items) - 1
        count = 0
        while i >= 0:
            if type(items[i]) == str:
                item = str(items[i])
                count += 1
            i-= 1
        return count
```

```
(d) def count_strings (items):
        count = 0
        for i in items:
            if int(i):
                count += 0
            else:
                count += 1
        return count
```

(17) You're writing a program to help organise a music library. You'd like to group songs based on the following:

- **short**: 60 seconds or less
- **medium**: Between 60 and 120 seconds, exclusive
- **long**: 120 seconds or more

Which of the following sets of values, representing song length, would be *most* useful for testing your program?

a) `[30, 90, 150, 180]`
[✓ b) `[-1, 59, 90, 120]`]
c) `[0, 30, 60, 90]`
d) `[29, 59, 89, 150]`

(19) Which line of code would cause an error to be thrown when this script is executed with the following inputs in this order: **Bob, 777**

```
1 name = input("Please enter your name: ")
2 number = input("Please enter your favourite number: ")
3 combined = name + number
4 combined = combined * 2
5 lucky_character = combined[12]
6 print(lucky_character)
```

[✓ a) 5]
b) 2
c) 3
d) 4

(19) You're writing a function to help organise a music library. It should return a recommended group based on the following (assume all song lengths are positive integers):

- **short**: 60 seconds or less
- **medium**: Between 60 and 120 seconds, exclusive
- **long**: 120 seconds or more

Which of the following implementations would have a logic error, based on the above problem description?

```
a)  def song_group(song_length):
        recommendation = "short"
        if song_length >= 120:
            recommendation = "long"
        if song_length > 60:
            recommendation = "medium"
        return recommendation
```

```
b)  def song_group(song_length):
        if song_length <= 60:
            return "short"
        elif song_length < 120:
            return "medium"
        else:
            return "long"
```

```
c)  def song_group(song_length):
        if song_length <= 60:
            return "short"
        if song_length < 120:
            return "medium"
        return "long"
```

```
D)  def song_group(song_length):
        if song_length >= 120:
            recommendation = "long"
        elif song_length > 60:
            recommendation = "medium"
        else:
            recommendation = "short"
        return recommendation
```

20. Which of the following lines of code will run but then give a runtime error?

a)
```
vals = [0, 2, 4]
print(vals[vals[0]])
```
b)
```
vals = [0, 2, 4]
print(vals[vals[1]])
```
c)
```
vals = [0, 2, 4]
print(vals[vals[2]])
```
d) They will all give a runtime error.

(21) Complete the following function by copying the code below into the answer box and then updating the function to **return** a single string created with words separated by space from the given tuple.

Example:

Input: `create_sentence(('hello', 'world', 'from', 'python'))`

returns: `'hello world from python'`

```
def create_sentence(words: tuple) -> str:
```

Question 22 is on the following page

(22) Write a program that accepts a list of positive integers as **command line arguments** and **prints** the number that has the **fewest digits**. You may assume that arguments provided are **numeric**. If there is a **tie**, print the **first** number found.

Sample input and output:

```
$ python3 program.py 123 71 9316 11
71
```

This is the end of Section A: Foundation Knowledge

Section B will begin on the following page.

Section B: Advanced Knowledge begins here. There are 6 total questions.

Complete if you are confident you have OK or above in the Foundation Test/Exam.

*All questions in this section (Q1-Q6) should be answered **only** in the space provided in this booklet (any answer outside the space provided will not be marked).*

(1) Implement a function called `valid_id()`. It should **use a regular expression** to verify if the string **begins with the following**:

- Four English letters from A to Y (so, excluding Z) using **any letter case**.
- After the four letters, it has 6 digits between 2 and 8 inclusive.

If the string meets the stated conditions, the function should return the boolean `True`, otherwise it should return the boolean `False`. For example, the following code would give the shown output:

```
print(valid_id("LWAR732258"))
print(valid_id("LwAr7322584X4Z")) # True, begins correctly.
print(valid_id("AB234567")) # False, not enough letters
print(valid_id("alez1234")) # False, contains z and 1
```

Outputs:

```
True
True
False
False
```

(2) Given the following definition for the class Song:

```
class Song:
    def __init__(self, title: str, length: int):
        self.title = title
        self.length = length

    def get_title(self):
        return self.title

    def get_length(self):
        return self.length
```

Complete the implementation for the special methods of the class `Album` below, so that the following code snippet would give the shown output:

```
s1 = Song("Imagine", 120)
s2 = Song("Royals", 90)
s3 = Song("Jump", 150)
album = Album([s1,s2,s3])
print(album)       # each song on a separate line
print(len(album))  # total length of all songs combined
```

Outputs:

```
Imagine, 120
Royals, 90
Jump, 150
360
```

```
class Album:
    def __init__(self, songs: list):


    def __repr__(self):


    def __len__(self):
```

(3) Your task is to generate a list of numbers, that begins with the values 0 and 1. The next number in the list is the sum of the previous two, and this continues until that sum exceeds a given threshold number (this last number that exceeds the threshold number is not included in the list). For example:

```
number_list = number_sequence(10)
print(number_list)
```

Outputs:

```
[0, 1, 1, 2, 3, 5, 8]
```

(a) Write a function called `number_sequence` below, to meet the criteria of the task defined above. You must implement this as a **non**-recursive function. i.e. you are not allowed to call the function `number_sequence`.

```
def number_sequence(          ):
```

(b) Now write a **recursive** version of the function called `recursive_sequence` below, to meet the criteria of the task defined earlier. You **must** implement this as a **recursive** function. i.e. you must call the function `recursive_sequence`.

```
def recursive_sequence(         ):
```

(4) Given the following definition for class Book:

```
class Book:
    def __init__(self, title: str, book_id: int):
        self.title = title
        self.id = book_id
```

The `Library` class should store a list of `Book` objects. There can be multiple libraries, but each new book added to any library should be given a unique sequential ID number (shared across all libraries), beginning at 0. If a book is deleted from a Library, it should be removed from the book list without changing the next ID number created. Finish the `Library` class below, so that the following code would give the shown output.

```
city_library = Library()
school_library = Library()
city_library.add_book("Harry Potter")
school_library.add_book("Dune")
school_library.delete_book(1)
city_library.add_book("Skyfire")
city_library.print_books()
```

Outputs:

```
Harry Potter, ID: 0
Skyfire, ID: 2
```

```
class Library:




  def __init__(self):
    self.books = []

  def add_book(self, title):









  def delete_book(self, id):











  def print_books(self):
    [print(f"{book.title}, ID: {book.id}") for book in self.books]
```

(5) Given the following definition for class Character:

```
class Character:
    def __init__(self, name, health):
        self.name = name
        self.health = health

    def attack(self):
        print(f"{self.name} attacks!")
```

Implement two **subclasses** of `Character` (`Warrior` and `Mage`) so that the following code would result in the output shown:

```
warrior = Warrior("Aragorn", 100, "a thousand suns!")
mage = Mage("Gandalf", 80, 120)
bystander = Character("Cabbage Man", 1)
characters = [warrior, mage, bystander]
for character in characters:
    character.attack()
```

Outputs:

```
Aragorn attacks with the strength of a thousand suns!
Gandalf casts a spell! Mana: 120
Cabbage Man attacks!
```

(6) Memes and inside jokes are a huge part of internet culture, but sometimes we fear things have gone too far.

Write a **decorator** called `safety` that can be used for functions that have a single parameter that you can assume is a string. This decorator should identify if the string being passed as input contains the text "`skibidi`", using **any letter casing**. If it does, then "`We don't do that here.`" should be printed and prevent the function from running. If it does not, then run the function as usual and return the expected output.

For example, if the function `print_story()` used the `safety` decorator, and simply called the `print()` function on the input string, then the following code would output:

```
print_story("Chicken Jockey")
print_story("what a skibidi rizzler")
print_story("stop trying to relate it's cringe")
print_story("the existential dread of the global climate")
print_story("and housing crisis is why i doom scroll")
```

Outputs:

```
Chicken Jockey
We don't do that here.
stop trying to relate it's cringe
the existential dread of the global climate
and housing crisis is why i doom scroll
```

First, write the **decorator:**

Now implement the `print_story()` function as it is defined above, ensure it uses the `safety` decorator.

END OF EXAMINATION

You may use the remaining blank pages as additional working space.

Please clearly mark each question you completing in this space.

Please clearly mark each question you completing in this space.

Please clearly mark each question you completing in this space.