Programming with Generative AI (2025): Practice Exam

Total points 31/50

Solutions will be discussed on: 31st October 2025
Time: 11:00AM to 12:00PM

Join Zoom Meeting: https://us06web.zoom.us/j/85987192386?pwd=LFIlPqFIuX1fiQ297tnXJPX7bOYeRs.1
Meeting ID: 859 8719 2386
Passcode: 139089

YouTube: https://youtube.com/live/IY0xTbMOclA?feature=share

Multiple Choice/Multiple Select Questions

14 of 20 points

Which of these is **NOT** a property of an algorithm? * 1/1

○ It has a finite number of steps

● Each step is performed exactly once

○ Each step runs in a finite amount of time

○ Each step can be translated into one or more steps in some programming language

Your friend performs the following experiment in the REPL: *1/1

```
>>> (3 // 2) == 1.5
False
```

Which of these is the **best** way to explain the result of this experiment to your friend?

○ The == operator checks if two objects are identical

○ The == operator checks if two objects have the same type

○ The float value 1.5 is an approximation of the real value 1.5

○ The expression (3 // 2) evaluates to an int whereas 1.5 is a float

● The expression (3 // 2) evaluates to an int whose value is NOT 1.5

○ The expression (3 // 2) evaluates to a float whose value is NOT 1.5

○ The // symbol is a comment, so Python ignores everything after that symbol

Consider this Python program: * 1/1

```
feeling = input('How are you feeling? ')  # Line 1
print(feeling)  # Line 2
```

Your friend runs this program and types:
"OK

What will your friend observe?

○ A syntax error is reported on Line 1

○ A syntax error is reported on Line 2

● The program runs correctly and prints "OK

○ The program runs correctly and prints "OK"

○ The program runs correctly and prints "OK within single quotes i.e., "'OK'

○ A run-time error occurs on Line 1 since your friend did not close the double-quote

Consider this Python program: * 1/1

```
age = input('What is your age? ')  # Line 1
print(int(age))  # Line 2
```

Your friend runs this program and types:
"55"

What will your friend observe?

○ A syntax error is reported on Line 1

○ A syntax error is reported on Line 2

○ A run-time error occurs on Line 1

● A run-time error occurs on Line 2

○ The program runs correctly and prints 55

○ The program runs correctly and prints "55"

Suppose you are asked to write the following Python function: *0/1

```
def is_good(password: str) -> bool:
    """Check if password is a good password.
    >>> is_good('123')
    False
    >>> is_good('^pT3l3P40nix!')
    True
    """
    pass
```

Which of the following will be the **most useful** helper function to define?

○ def length(password: str) -> int:

○ def strength(password: str) -> int:

● def is_strong(password: str) -> bool:

○ def has_digits(password: str) -> bool:

Which of these are examples of using increasing computing power to assist programmers? Select **all** correct examples from the options below. *1/1

☑ Providing programmers with tools to find errors in their code (e.g., MyPy)

□ Allowing programmers to write code using simple programs (e.g., Notepad)

□ Helping programmers to manually write code in machine language more easily

☑ Providing programmers with tools to visualize code execution (e.g., PythonTutor)

Your friend initializes a variable **n** with some unknown value and then performs the following experiment in the REPL: *1/1

```
>>> 2 * (n / 2) + (n % 2) == n
False
```

Select **all** values that **n** can have from the options below.

□ 0

☑ 1

□ 2

☑ -1

□ -2

Your friend initializes a variable **x** with some unknown value and then performs the following experiment in the REPL: *1/1

```
>>> x + x + x == 2 * x
True
```

Select **all** values that **x** can have from the options below.

☑ 0

☑ 0.0

□ float('nan')

☑ float('inf')

☑ float('-inf')

☑ [] # empty list

☑ " # empty string

□ None # the special Python object None

Consider this sequence of Python statements: *1/1

```
x = -1
y = x
x //= 2
```

After performing these statements, select **all** expressions that evaluate to **True** among these options:

☐ x > y

☐ x < y

☑ x == y

☑ x is y

☐ 2 * x == x

☐ 2 * x == y

☐ 2 * y == x

☑ type(x) == type(y)

Consider the following **poorly-designed** Python function: *0/1

```
def first(n: int):  # Line 1
   return int(str(abs(n))[-1])  # Line 2
```

Select **all** changes among these options that will improve the code:

☑ (Line 1) Rename the function to last

□ (Line 1) Rename the function to firstDigit

□ (Line 1) Rename the function to units_digit

□ (Line 1) Add the return type-hint: -> str

☑ (Line 1) Add the return type-hint: -> int

☑ (Between Line 1 and Line 2) Add a docstring explaining how Line 2 works

☑ (Between Line 1 and Line 2) Add doctests illustrating how the function works

Consider the following Python function: *1/1

```
def mystery(s: str, a: int) -> bool:
    a = str(a)
    if len(s) <= len(a):
        return False
    else:
        if a in s:
            return True
        else:
            return False
```

Your friend has tried to simplify this function as follows:

```
def mystery(s: str, a: int) -> bool:
    return str(a) in s
```

On which of these inputs will the **modified** function return a **different** value?

□ s = '2', a = -2

□ s = '-2', a = 2

☑ s = '123', a = 123

□ s = 'abc', a = 123

□ s = '1a2b3c', a = 123

□ s = '123abc', a = 123

□ None of these

Consider the following Python function: *1/1

```
def mystery(s: str) -> bool:
   if not s:
      return False
   elif s.count('0')/len(s) > 0.5:
      return True
   else:
      return False
```

Select **ALL** appropriate values of **EXPRESSION** so that the above function can be rewritten as:

```
def mystery(s: str) -> bool:
   return EXPRESSION
```

☐ min(s.count('0')/len(s), 0) > 0.5

☐ max(s.count('0')/len(s), 0) > 0.5

☐ s or (s.count('0')/len(s) > 0.5)

☐ (s.count('0')/len(s) > 0.5) or s

☑ s and (s.count('0')/len(s) > 0.5)

☐ (s.count('0')/len(s) > 0.5) and s

☐ len(s) or (s.count('0')/len(s) > 0.5)

☐ (s.count('0')/len(s) > 0.5) or len(s)

☑ len(s) and (s.count('0')/len(s) > 0.5)

☐ (s.count('0')/len(s) > 0.5) and len(s)

□ None of these

Your friend is translating the following Python function into C: *0/1

```
def max_count(data: list[int], min_val: int) -> int:  # Line 1
    """Find the count of the largest value in data
    whose value is at least min_val."""
    largest = None  # Line 2
    count = 0  # Line 3
    for item in data:  # Line 4
        if item >= min_value:  # Line 5
            if count == 0 or item > largest:  # Line 6
                largest = item  # Line 7
                count = 1  # Line 8
            elif item == largest:  # Line 9
                count += 1  # Line 10
    return count  # Line 11
```

What points should your friend keep in mind?

☑ (Line 1) The C function will have two parameters

☑ (Line 1) The list parameter (data) can be a const array in C

□ (Line 2) This can be translated in C as: int largest; // declared, not initialized

□ (Line 2) Since C has no equivalent of None, this line cannot be translated directly

☑ (Line 4) The for-loop can be translated into C using a while-loop

☑ (Line 5) In C, the if-condition MUST be written within parentheses (...)

□ (Line 6) The || operator in C has a similar short-circuit evaluation logic to Python's "or" operator

□ (Line 6) In C, we have to use = instead of == within if-conditions

□ (Lines 7 and 8) The body of the if MUST be within curly brackets { ... }

☑ (Line 9) "elif" must be translated as "else if" in C

☑ (Line 10) The body of the translated elif MUST be within curly brackets { ... }

□ (Line 10) The translation can use C's ++ operator

Which is the **most appropriate** name for this Python function? *1/1

```
def f(a: int, b: int, c: int) -> int:
  total = a + b + c
  return total - min(a, b, c) - max(a, b, c)
```

○ mean

● median

○ mode

○ None of these

Suppose you are asked to write the following Python function: *1/1

```
def same(fahrenheit: float, celcius: float) -> bool:
    """Check if  fahrenheit and celcius
    represent the same temperature.
    >>> same(32.0, 0.0)
    True
    >>> same(0.0, 32.0)
    False
    """
    pass
```

Which of the following will be the **most useful** helper function to define?

● def to_celsius(temp_fahrenheit: float) -> float:

○ def to_fahrenheit(temp_fahrenheit: float) -> float:

○ def convert(fahrenheit: float, celsius: float) -> float:

○ def difference(fahrenheit: float, celsius: float) -> float:

Your friend has written the following Python function: * 0/1

```
def convert(s: str) -> str:
    """Convert all commas in s to spaces.
    >>> convert('1,2,3')  # doctest 1
    '1 2 3'
    >>> convert('1 2')  # doctest 2
    '1 2'
    """
    i = 0  # Line 1
    while i < len(s):  # Line 2
        if s[i] == ',':  # Line 3
            s[i] = ' '  # Line 4
        return s  # Line 5
```

Which of these indicates an error in the above code?

○ A run-time error on Line 1 for doctest 1

○ A run-time error on Line 2 for doctest 1

○ A run-time error on Line 3 for doctest 1

● A run-time error on Line 4 for doctest 1

○ The function does not return for doctest 1

○ The above function returns the wrong answer for doctest 1

○ A run-time error on Line 3 for doctest 2

○ A run-time error on Line 4 for doctest 2

○ The function does not return for doctest 2

☐ The above function returns the wrong answer for doctest 2

The following AI-suggested code makes several assumptions: *1/1

```
def avg_letter_ord(s: str) -> float:
    """Calculate the average ordinal value
    of alphabet letters in s."""
    return sum(ord(c) for c in s if c.isalpha()) / len(s))  # AI suggestion
```

For which of these function calls will it be helpful to clarify the expected behaviour by "asking the client"?

☑ avg_letter_ord('') # empty string

☐ avg_letter_ord('abc')

☑ avg_letter_ord('123')

☑ avg_letter_ord('über')

The following AI-suggested code makes several assumptions: *1/1

```
def next_day(day: str) -> str:
    """Return the day that follows the given day.
    >>> next_day('Monday')
    'Tuesday'
    """
    # AI suggestion
    days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday',
'Friday', 'Saturday', 'Sunday']
    if day not in days:
        raise ValueError("Invalid day provided.")
    index = days.index(day)
    next_index = (index + 1) % len(days)
    return days[next_index]
```

For which of these function calls will it be helpful to clarify the expected behaviour by "asking the client"?

☑ next_day('monday')

□ next_day('Tuesday')

☑ next_day('Tues')

☑ next_day('day')

A Gen AI tool has written the following recursive Python function: *0/1

```
def rec_min_factor(n: int, factor: list[int]) -> int:
    """Find the smallest integer in factor
    that is a factor of n, using recursion.
    Return n + 1 if no factor exists.
    >>> rec_min_factor(10, [5, 2, 3])  # doctest 1
    2
    >>> rec_min_factor(5, [2, 6, 3])  # doctest 2
    6
    """
    if not factor:
        return n + 1
    first = factor[0]
    rest = factor[1:]
    if n % first == 0:
        return min(first, rec_min_factor(n, rest))
    else:
        return rec_min_factor(n, rest)
```

Select **all** true statements from the options below.

□ doctest 1 is invalid

□ The code passes doctest 1

☑ doctest 2 is invalid

□ The code passes doctest 2

□ On some input(s), there is a ZeroDivisionError

□ On some input(s), we get infinite recursion

Your friend has used a popular Large Language Model (LLM) to generate test cases for the following C function: *0/1

```
double to_celsius(double fahrenheit) {
  // Translate Fahrenheit to Celsius.
  // Return NaN if fahrenheit is NaN or
  // it is below absolute zero.
}
```

Your friend observes that many of the LLM-generated test cases are invalid. Select all **likely** explanations for this observation.

□ The LLM does not know how to write test cases in C

☑ The LLM hallucinates the value of absolute zero on the Fahrenheit scale

☑ The LLM has not been trained on some of the relevant concepts (Fahrenheit, Celsius, NaN, absolute zero)

□ None of these

Code comprehension 2 of 3 points

Your friend has written the following Python function:

```
def check(data: list[str], x: str) -> bool:
  for item in data:
    if item in x:
      return False
```

return True

Select **all** function calls that return **True** * 1/1

☑ check([ ], '') # data = empty list, x = empty string

□ check([''], '') # data = list containing empty string, x = empty string

□ check(['a', 'b', 'c'], 'abc')

☑ check(['a', 'b', 'c'], 'xyz')

☑ check(['abc'], 'axbc')

☑ check(['axbc'], 'abc')

Which of these is the **most appropriate** docstring for the function above? *1/1

○ Check if every string in data is a substring of x

○ Check if x is a substring of every string in data

● Check if no string in data is a substring of x

○ Check if x is NOT a substring of any string in data

The body of the above function can be written as a single statement: *0/1

return sum(1 for item in data if **EXPRESSION1**) == **EXPRESSION2**

What are **all possible** values of **EXPRESSION1** and **EXPRESSION2**?

- ☑ EXPRESSION1 is: item in x
- □ EXPRESSION1 is: x in item
- □ EXPRESSION1 is: item not in x
- □ EXPRESSION1 is: x not in item
- □ EXPRESSION2 is: len(x)
- ☑ EXPRESSION2 is: len(data)
- □ EXPRESSION2 is: len(item)

Code comprehension 1 of 3 points

Your friend has written the following Python function:

```
def max_count(data: list[int]) -> int:
  """Count the number of times the
  maximum int appears in data."""
  m = 0  # Line 1
  count = 0  # Line 2
  i = 0  # Line 3
  while i < len(data):  # Line 4
    if data[i] > data[m]:  # Line 5
      m = i  # Line 6
```

```
            count = 1  # Line 7
        if data[i] == data[m]:  # Line 8
            count += 1  # Line 9
        i += 1  # Line 10
    return count
```

Which of the following are true? * 0/1

☑ max_count([ ]) == 0

☐ max_count([-5]) == 0

☑ max_count([-5]) == 1

☑ max_count([1, 2, 1]) == 1

☐ max_count([1, 2, 1]) == 2

☐ max_count([2, 1, 2]) == 1

☑ max_count([2, 1, 2]) == 2

☐ The while-loop uses Pattern 1: Iterate until success

☐ The while-loop uses Pattern 2: Accumulate

Which of the following are true for **some** values of **data**? * 1/1

☐ max_count(data) gets stuck in an infinite loop

☐ max_count(data) fails on Line 5

☐ max_count(data) fails on Line 8

☑ max_count(data) == len(data)

Your friend's code is **buggy**. Which of the following changes are necessary to fix the code? *0/1

☐ (Line 1) Rewrite as: m = None

☐ (Line 2) Rewrite as: count = 1

☐ (Line 3) Rewrite as: i = 1

☐ (Line 5) Rewrite as: if data[i] > m:

☑ (Line 5) Rewrite as: if data[i] >= data[m]:

☑ (Line 8) Replace if with elif

☐ (Line 10) Indent to the same level as Line 9

Code comprehension

0 of 3 points

Your friend has written the following Python function:

```
def process(data: list[list[int]]) -> list[int]:
  result = [ ]  # Line 1
  for inner in data:  # Line 2
    for item in inner:  # Line 3
      if item % 2:  # Line 4
        result.extend([item])  # Line 5
  return result
```

What is the result of calling **process([[1, 2, 5], [3, 4, 0]])**? * 0/1

Write your answer within square brackets. If the list has multiple values, separate these using commas and spaces.

Examples of **valid** answers: [] or [45] or [6, 7]

Examples of **invalid** answers: [6 7] or (6, 7) or {6, 7}

[2,4,0]

For any input **data: list[list[int]]**, select **all** expressions below that always evaluate to True. *0/1

☐ len(process(data)) == len(data)

☐ process(2 * data) == 2 * process(data)

☐ process(data[::-1]) == process(data)[::-1]

☑ process(data) in data

Identify **all** changes that will improve the performance and/or readability of the above code *0/1

☐ (Line 1) Rewrite as: result = ( ) # empty tuple

☑ (Line 2 and Line 3) Combine into a single loop: for item in zip(data, inner):

☑ (Line 4) Rewrite as: if item % 2 == 0:

☐ (Line 5) Rewrite as: result += item

☑ (Line 5) Rewrite as: result.append(item)

☐ (Line 5) Rewrite as: result = result + [item]

Code writing 4 of 4 points

Your friend has written this Python function to translate words in US spelling to Indian spelling. The doctests specify how to handle upper vs. lower case, and illegal inputs.

```
def to_ind_lower(word: str) -> str:
    """
    >>> to_ind_lower('Analyze')
    'analyse'
    >>> to_ind_lower('BEE')
    'bee'
    """
    to_ind = {'analyze': 'analyse', 'behavior': 'behaviour', 'color': 'colour'}
    result = BLANK1
    if BLANK2:
        result = BLANK3
    return BLANK4
```

The **most appropriate** choice for **BLANK1** is: * 1/1

○ word.lower

● word.lower()

○ to_ind[word]

○ to_ind_lower(word)

The **most appropriate** choice for **BLANK2** is: * 1/1

○ to_ind[word]

○ to_ind[result]

○ word in to_ind

● result in to_ind

○ result in to_ind.items()

The **most appropriate** choice for **BLANK3** is: * 1/1

○ to_ind[word]

● to_ind[result]

○ to_ind[word].lower()

○ to_ind[result].lower()

The **most appropriate** choice for **BLANK4** is: * 1/1

● result

○ result.lower()

○ to_ind[word]

○ to_ind[result]

○ to_ind[word].lower()

Code writing 5 of 5 points

Your friend has written this Python function to find the maximum profit that can be earned by buying a stock on day **i** at price **cost[i]** and selling it at a later day **j** at price **cost[j]**. Non-positive values of **cost** should be ignored. The doctests specify how to handle situations where no profit can be earned.

def max_profit(cost: list[int]) -> int:

```
"""
>>> max_profit([3])
0
>>> max_profit([2, 3, -1, 2, 3])
1
"""
result = BLANK1
for i in range(BLANK2):
    for j in range(BLANK3, BLANK4):
        if BLANK5:
            result = cost[j] - cost[i]
return result
```

The **most appropriate** choice for **BLANK1** is: * 1/1

● 0

○ cost[0]

○ min(cost)

○ max(cost)

The **most appropriate** choice for **BLANK2** is: * 1/1

○ cost

○ result

○ len(result)

● len(cost) - 1

○ len(cost - 1)

The **most appropriate** choice for **BLANK3** is: * 1/1

○ 0

○ 1

○ i

● i + 1

The **most appropriate** choice for **BLANK4** is: * 1/1

○ cost

○ result

● len(cost)

○ len(result)

○ len(cost) - 1

The **most appropriate** choice for **BLANK5** is: * 1/1

○ 0 < cost[i] < cost[j]

○ cost[j] - cost[i] > result

● cost[j] - cost[i] > result and cost[i] > 0

○ max(cost[j] - cost[i] - result, cost[i]) > 0

Task comprehension 3 of 3 points

Consider the following problem:

```
def merge(x: list[int], y: list[int]) -> list[int]:
  """Return a list containing only the unique
  integers that appear in list x + y (i.e., in the
  combined list). In the resulting list, all values
  that appear in list x must be before values that
```

appear only in list y. Values that appear in list x must be in ascending order in the resulting list. Similarly, values that appear only in list y must be in ascending order in the resulting list."""

Complete the following doctest by writing **only** the value returned by the function on this input: *1/1

>>> merge([1, 5, 3], [4, 2])

Write your answer within square brackets. If the list has multiple values, separate these using commas and spaces.

Examples of **valid** answers: [] or [45] or [6, 7]

Examples of **invalid** answers: [6 7] or (6, 7) or {6, 7}

[1,3,5,2,4]

Complete the following doctest by writing **only** the value returned by the function on this input: *1/1

>>> merge([1, 3, 1], [3, 2, 4])

Write your answer within square brackets. If the list has multiple values, separate these using commas and spaces.

Examples of **valid** answers: [] or [45] or [6, 7]

Examples of **invalid** answers: [6 7] or (6, 7) or {6, 7}

[1,3,2,4]

Complete the following doctest by writing **only** the value returned by the function on this input: *1/1

>>> merge([3, 0, 1, 0], [2, 1, 3, 2])

Write your answer within square brackets. If the list has multiple values, separate these using commas and spaces.

Examples of **valid** answers: [] or [45] or [6, 7]

Examples of **invalid** answers: [6 7] or (6, 7) or {6, 7}

[0,1,3,2]

Code rewriting

1 of 2 points

Consider the following Python function:

```
def all_indices(data: list[int], lo: int, hi: int) -> list[int]:
    """Return a list of all indices in data (in ascending
    order) that the integer x at index i satisfies: lo < x < hi."""
    result = [ ]  # Line 1
    i = 0  # Line 2
    while i < len(data):  # Line 3
        if lo < data[i] < hi:  # Line 4
            result = result + [i]  # Line 5
        i += 1  # Line 6
    return result
```

Your friend wants to rewrite this function using a for-loop by rewriting line 3 as: *1/1

for i, item in enumerate(data):

In addition to this, what else should be changed?

☐ Delete Line 1

☑ Delete Line 2

☑ Replace Line 4 with: if lo < item < hi:

☑ Replace Line 5 with: result.append(i)

☐ Replace Line 5 with: result.append(item)

☐ Replace Line 5 with: result += [item]

☐ Replace Line 5 with: result = result + [item]

☑ Delete Line 6

The entire body of the **all_indices** function can be rewritten as: *0/1

return EXPRESSION

where **EXPRESSION** is

● [i for i, item in enumerate(data) if lo < i < hi]

○ [item for i, item in enumerate(data) if lo < i < hi]

○ [i for i, item in enumerate(data) if lo < item < hi]

○ [item for i, item in enumerate(data) if lo < item < hi]

Refute 1 of 7 points

The following recursive Python function is **buggy**:

```
def min_word(s: str) -> str:
    """Find the lexicographically smallest
    word (case sensitive) in sentence s.
    >>> min_word('')  # doctest 1
    ''
    >>> min_word('all able')  # doctest 2
    'able'
    >>> min_word('All able')  # doctest 3
    'All'
    """
    try:
        words = s.split(' ')  # split on single spaces  # Line 1
        if len(words) > 1:  # Line 2
            rest_index = s.index(words[1])  # Line 3
            return min(words[0], min_word(s[rest_index : ]))  # Line 4
        return words[0]  # Line 5
    except:
        return ''  # empty string  # Line 6
```

The code passes **doctest 1** because it returns the empty string on Line number: *0/1

6

Code tracing

The code passes **doctest 2** because:
(Line 1) The value of **words** is **BLANK1**
(Line 2) The **if**-condition is **True**
(Line 3) **rest_index** is **BLANK2**
(Line 4) A recursive call is made with argument **'able'**
(Line 1) The value of **words** is **BLANK3**
(Line 2) The **if**-condition is **False**
(Line 5) The recursive call returns the value **BLANK4**
(Line 4) The function returns the value **'able'**

What is **BLANK1**? * 1/1

Write your answer within square brackets. If the list has multiple values, separate these using commas and spaces. Write strings within **single** quotes.

Examples of **valid** answers: [] or ['abc'] or ['a', 'b', 'c']

Examples of **invalid** answers: ['a' 'b'] or ["a", "b"] or ('a', 'b')

['all','able']

What is **BLANK2**? * 0/1

1

What is **BLANK3**? * 0/1

Write your answer within square brackets. If the list has multiple values, separate these using commas and spaces. Write strings within **single** quotes.

Examples of **valid** answers: [] or ['abc'] or ['a', 'b', 'c']

Examples of **invalid** answers: ['a' 'b'] or ["a", "b"] or ('a', 'b')

' '

What is **BLANK4**? * 0/1

Write strings within single quotes.

able

The **shortest** string starting with the letter **'a'** on which the above code does **not** return the expected answer is: *0/1

Write strings within single quotes.

a

Instead of returning **'a'** on this input, the function incorrectly returns: *0/1

Write strings within single quotes.

" "