CMPT 120-D400 Mini Midterm, Fall 2024

This is a **20 minute closed book exam**: notes, books, computers, calculators, electronic devices, etc. are **not** permitted.

Semester Names

a) (6 marks) Write a function called `semester_name(code)` that returns the semester name for that code number. If an invalid code is given, then return `'invalid'`. The semester codes are summarized in this table:

Codes	Semester
1, 2, 3	Fall
4, 6, 7, 8	Spring
5, 9	Summer

Examples

```
>>> semester_name(2)
'Fall'
>>> semester_name(3)
'Fall'
>>> semester_name(7)
'Spring'
>>> semester_name(5)
'Spring'
>>> semester_name(11)
'invalid'
```

Solution

```
def semester_name(code):                # 1 mark for correct header
    if 1 <= code <= 3:                  # 1 mark for Fall case
        return 'Fall'
    elif code == 4 or 6 <= code <= 8: # 1 mark for Winter case
        return 'Winter'
    elif code == 5 or code == 9:        # 1 mark for Spring case
        return 'Spring'
    else:
        return 'invalid'          # 1 mark for invalid case

# 1 mark for overall correct and consistent indentation
# Up to -1 mark for code that does unnecessary work, or uses
#               unnecessary memory, or is very hard to read
```

semesters.txt

```
9
4
4
-3
2
15
```

b) (7 marks) The text file `semesters.txt` contains one or more integer code numbers, one number per line. An example is in the box on the right. Write a program that reads this file and prints a count of how many of the codes are valid semesters. For example, for `semesters.txt` your program should print:

```
4 valid semesters
```

Of course, your program should work with any numbers in `semesters.txt`, not just the ones shown. You can re-use `semester_code` from above if you like

Solution

```
file_obj = open('semesters.txt')              # 1 mark correctly opening file
valid_count = 0                               # 1 mark initializing count
for line in file_obj:                         # 1 mark looping through file lines
    if semester_name(int(line)) != 'invalid':  # 2 marks correct if condition,
        valid_count += 1                      #         uses int, increments count

print(f'{valid_count} valid semesters')       # 1 mark prints correct final message

# 1 mark for overall correct and consistent indentation
# Up to -1 mark for code that does unnecessary work, or uses
#               unnecessary memory, or is very hard to read
```

Multiple Choice

For each of the following questions, fill in **the one best answer** on the answer sheet. Every correct answer is worth 1 mark. Incorrect answers, unanswered questions, questions with more than one answer, or questions with illegible answers, are worth 0.

1) Python:

i) is a popular in data science and machine learning
ii) has a reputation for making extremely fast-running programs

A. i) and ii) are both true
B. i) and ii) are both false
C. i) is false and ii) is true
D. i) is true and ii) is false

2) A Python variable name can:

i) start with a digit
ii) contain a period character

A. i) and ii) are both true
B. i) and ii) are both false
C. i) is false and ii) is true
D. i) is true and ii) is false

3) What is the *last* number this prints?

```
i = 100
while i > 0:
    print(i)
    i -= 1
```

A. 1
B. 0
C. -1
D. -2
E. some other number

4) What is the *last* number this prints?

```
i = 100
while i > 0:
    i -= 1
    print(i)
```

A. 1
B. 0
C. -1
D. -2

5) How many *s does this print?

```
j = 10
while j >= 2:
    print('*')
    j -= 2
```

A. 3
B. 4
C. 5
D. 6

6) There is an `int` that can replace `???` to make this an infinite loop.

```
i = ???
while i > 0:
    print(i)
    i -= 1
```

A. true
B. false

7) i) Any for-loop that uses `range` can be re-written as an equivalent while-loop.
ii) Any while-loop can be re-written as an equivalent for-loop that uses `range`.

A. i) and ii) are both true
B. i) and ii) are both false
C. i) is false and ii) is true
D. i) is true and ii) is false

8) What does this print?

```
a = 1
b = 2
a = b
b = a
print(a, b)
```

A. `1 2`
B. `2 1`
C. `2 2`
D. `4 4`

9) What does this print?

```
print = 'print' # line 1
print(print)    # line 2
```

A. `<built-in function print>`
B. `print`
C. nothing, line 1 causes an error
D. nothing, line 2 causes an error