UNIVERSITY OF TORONTO
Faculty of Arts & Science

Winter 2023 Term Test 2

CSC 108 H1S

Duration: 50 minutes

Aids Allowed: None

*Do **not** turn this page until you have received the signal to start.*
In the meantime, write your name, student number, and UTORid below (please do this now!) and *carefully* read *all* the information on the rest of this page.

First (Given) Name(s):

Last (Family) Name(s):

10-Digit Student Number:

UTORid (e.g., `pitfra12`):

UToronto Email address (e.g., `firstname.lastname@mail.utoronto.ca`):

- This test consists of 5 questions on 12 pages (including this one), printed on both sides of the paper. *When you receive the signal to start, please make sure that your copy of the test is complete.*
- Answer each question directly on the test paper, in the space provided.

Marking Guide

Nº 1:	______/ 4
Nº 2:	______/ 4
Nº 3:	______/ 4
Nº 4:	______/ 2
Nº 5:	______/ 4
TOTAL:	______/18

Question 1. [4 marks]

Assume each of the blocks of code below are entered into the Python Shell. Each Part is independent of the others. In each box, write what would be printed in the Python Shell. If the code would cause an error, write ERROR in the box.

Part (a) [1 mark]

```
>>> L1 = [1, 2, 3]
>>> L2 = L1[:]
>>> L3 = L1
>>> L1.append(99)
>>> print(L2)
```

```
>>> print(L3)
```

Part (b) [1 mark]

```
>>> nums = [[1, 2], [3, 4], [5, 6]]
>>> print(len(nums))
```

```
>>> nums[0] = 'Yay'
>>> print(nums)
```

Part (c) [2 marks]

```
>>> colours = {"Bumbly": "white", "Mia": "grey", "Chirly": "green"}
>>> print(colours["Bumbly"])
```

```
>>> colours["Mia"] = "black"
>>> print(len(colours))
```

```
>>> print("green" in colours)
```

```
>>> print(colours[0])
```

Question 2. [4 marks]

Finish the docstring examples, fill in the type contract (be **specific** about the types – e.g. for a tuple of strings you must write `tuple[str]` rather than just `tuple`), and write a good docstring description for the function below.

```
def mystery(lst: [          ], n1: [      ], n2: [      ]) -> [          ]:
    """
    [                                                    ]

    >>> L = [[1, 2, 3], [2, 45, 22]]
    >>> mystery(L, 2, 99)
    >>> L
    [                                          ]

    >>> L = [[1, 2, 3], [2, 45, 22]]
    >>> mystery(L, 20, 99)
    >>> L
    [                                          ]

    """

    for i in range(len(lst)):
        for j in range(len(lst[i])):
            if lst[i][j] == n1:
                lst[i][j] = n2
```

Question 3. [4 marks]

Bob has learned about Python dictionaries now and wants to revisit the function from A2 that builds a numerology list.

Recall that, on Assignment 2, we used `NUM_COMPATIBILITY_DATA`, a list of strings, to represent number compatibility. Each string indicated compatibility between a pair of numbers. For example, `'1,2,YES'` indicates that `2` is a compatible num for `1`.

Instead of this string representation, for this question we will represent compatibility with a list of the form `[N1, N2, BOOL]` where `BOOL` is either `True` or `False`. If `True` then `N2` is a compatible num for `N1` (but not necessarily the other way around, just like in the assignment). If `False`, `N2` is not a compatible num (i.e. an incompatible num) for `N1`.

On the following page, help Bob finish the function body according to the provided docstring.

This space left intentionally blank (except for this sentence). [You may use the space below for rough work if you need. This page will NOT be marked.]

```
def build_numerology_dict(data: list[list]) -> dict[int, tuple[list[int]]]:
    """
    Given compatibility `data` where each element is of the form
    [N1, N2, BOOL], build and return a dictionary with the following
    structure:

        {
            n1: ([all compatible nums for n1], [all incompatible nums for n1]),
            n2: ([all compatible nums for n2], [all incompatible nums for n2]),
            ...
        }

    >>> test_list = [[1, 2, True], [1, 1, True], [1, 4, False], [2, 3, True],
                      [3, 1, True], [3, 2, False]]
    >>> build_numerology_dict(test_list)
    {1: ([2, 1], [4]), 2: ([3], []), 3: ([1], [2])}
    """

    d = {}

    for sublst in data:
        n = sublst[0]
        m = sublst[1]
        compatible = sublst[2]
        # Update the dictionary as required
        # Hint: you may initialize a value to ([], []) - this is a tuple with 2 empty lists

    return d
```

Question 4. [2 marks]

We have a file called `pets.txt` that has the following content:

```
cat
dog
meow
woof
```

Write what each of the following code segments print.

If nothing is printed, write NO OUTPUT.

```
>>> f = open('pets.txt', 'r')
>>> x = f.read()
>>> print(f.readline().strip())
```

```
>>> f = open('pets.txt', 'r')
>>> x = f.readline()
>>> for line in f:
...     print(line.strip())
```

Question 5. [4 marks]

We will be dealing with a file that has a list of people's names, separated into groups using a `GROUP #` line as a divider.

An example of such a file, called `sample_groups.txt`, is below:

```
GROUP 1
sadia
paul
GROUP 2
fernando
sophia
tom
GROUP 3
yun
```

On the following page, fill in the blank boxes with the appropriate code to complete the function according to the docstring. Do **not** add in any additional lines or cross anything out.

This space left intentionally blank (except for this sentence). [You may use the space below for rough work if you need. This page will NOT be marked.]

```
def count_members(file: TextIO) -> list[int]:
    """Given an open file containing group information as described above,
    return a list containing the number of people in each group.

    Every group begins with a GROUP # line (where # is the number for that group).
    All member names contain only lower-case letters.

    Precondition:
    - the file has at least one group

    >>> file = open('sample_groups.txt')
    >>> count_members(file)
    [2, 3, 1]
    """
    counts = []
    line = file.readline().strip()
    while [                                        ]:
        group_count = [                            ]
        line = [                                   ]
        while [             ] and 'GROUP' not in [             ]

            [                                        ]

        counts.append([                            ])

    return counts
```

UNIVERSITY OF TORONTO
Faculty of Arts & Science

Winter 2023 Term Test 2

Short Python function/method descriptions:

```
__builtins__:
  int(x: object) -> int
    Convert x to an integer, if possible. A floating point argument will be truncated towards zero.
  len(x: object) -> int
    Return the length of list, tuple, or string x.
  list(iterable: object) -> object
    Return a list containing the items in iterable.
  min(a, b, c, ...) -> object
      With a single iterable argument, return its smallest item.
      With two or more arguments, return the smallest argument.
  open(name: str[, mode: str]) -> TextIO
    Open a file.  Legal modes are "r" (read) (default), "w" (write), and "a" (append).
    print(value: object) -> None
    Prints the value.
  range([start: int], stop: int, [step: int]) -> list-like-object of int
    Return the integers from start (inclusive) to stop (exclusive) with step
    specifying the amount to increment (or decrement).  If start is not specified,
    the sequence starts at 0.  If step is not specified, the values are incremented by 1.
  str(x: object) -> str
    Return an object converted to its string representation, if possible.
  tuple(iterable: object) -> object
    Return a tuple containing the items in iterable.
  type(x: object) -> the object's type
    Return the type of the object x.

file open for reading (TextIO):
  F.close() -> None
    Close the file.
  F.read() -> str
    Read until EOF (End Of File) is reached, and return as a string.
  F.readline() -> str
    Read and return the next line from the file, as a string. Retain any newline.
    Return an empty string at EOF (End Of File).
  F.readlines() -> List[str]
    Return a list of the lines from the file.  Each string retains any newline.

file open for writing (TextIO):
  F.close() -> None
    Close the file.
  F.write(x: str) -> int
    Write the string x to F and return the number of characters written.

list:
  x in L -> bool
    Produce True if and only if object x is in list L
  L.append(item: object) -> None
    Append item to end of list L.
  L.extend(items: iterable) -> None
    Extend list L by appending elements from items. Strings and lists are iterables whose elements
    are characters and list items respectively.
```

```
str:
  x in s -> bool
    Produce True if and only if string x is in string s.
  S.count(sub: str[, start: int[, end: int]]) -> int
    Return the number of non-overlapping occurrences of substring sub in string S[start:end].
    Optional arguments start and end are interpreted as in slice notation.
  S.find(sub: str[,i: int]) -> int
    Return the lowest index in S (starting at S[i], if i is given) where the
    string sub is found or -1 if sub does not occur in S.
  S.isalpha() -> bool
    Return True if and only if all characters in S are alphabetic
    and there is at least one character in S.
  S.isalnum() -> bool
    Return True if and only if all characters in S are alphanumeric
    and there is at least one character is S.
  S.isdigit() -> bool
    Return True if and only if all characters in S are digits
    and there is at least one character in S.
  S.islower() -> bool
    Return True if and only if all cased characters in S are lowercase
    and there is at least one cased character in S.
  S.isupper() -> bool
    Return True if and only if all cased characters in S are uppercase
    and there is at least one cased character in S.
  S.lower() -> str
    Return a copy of the string S converted to lowercase.
  S.replace(old: str, new: str) -> str
    Return a copy of string S with all occurrences of the string old replaced with the string new.
    S.split([sep: str]) -> List[str]
    Return a list that results from splitting S into substrings sep as the separator.
    Use any whitespace string as separator if sep is not specified.
  S.split([sep: str]) -> List[str]
    Return a list that results from splitting S into substrings sep as the separator.
    Use any whitespace string as separator if sep is not specified.
  S.startswith(S2: str) -> bool
    Return True if S starts with S2 and False otherwise.
  S.strip([chars: str]) -> str
    Return a copy of S with leading and trailing whitespace removed.
    If chars is given and not None, remove characters in chars instead.
  S.upper() -> str
    Return a copy of the string S converted to uppercase.

dict:
  D[k] --> object
    Return the value associated with the key k in D.
  del D[k]
    Remove the key-value pair k, D[k] from D.
  k in D --> bool
    Return True if k is a key in D and False otherwise.
  D.get(k: object) -> object
    Return D[k] if k in D, and None otherwise.
  D.keys() -> list-like-object of object
    Return the keys of D.
  D.values() -> list-like-object of object
    Return the values of D.
  D.items() -> list-like-object of Tuple[object, object]
    Return the (key, value) pairs of D, as 2-tuples.
```

[Use the space below for rough work. This page will not be marked unless you clearly indicate the part of your work that you want us to mark.]

[Use the space below for rough work. This page will not be marked unless you clearly indicate the part of your work that you want us to mark.]