PLEASE HAND IN

PLEASE HAND IN

UNIVERSITY OF TORONTO
Faculty of Arts and Science

AUGUST 2019 EXAMINATIONS

CSC 108 H1Y
Instructor:
Mark Kazakevich

Duration—3 hours

No Aids Allowed

You must earn at least 29 out of 71 marks (40%) on this final examination in order to pass the course. Otherwise, your final course grade will be no higher than 47%.

Student Number: _____ **UTORid:** _____

Family Name(s): _____ **First Name(s):** _____

---

*Do **not** turn this page until you have received the signal to start.*
In the meantime, please read the instructions below *carefully*.

This Final Examination paper consists of 10 questions on 21 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 paper is complete and fill in your Student Number, UTORid and Name above.*

- Comments and docstrings are not required except where indicated, although they may help us mark your answers.
- You do not need to put `import` statements in your answers.
- No error checking is required: assume all function arguments have the correct type and meet any preconditions.
- If you use any space for rough work, indicate clearly what you want marked.
- Do not remove pages or take the exam apart.
- As a student, you help create a fair and inclusive writing environment. If you possess an unauthorized aid during an exam, you may be charged with an academic offence.

MARKING GUIDE

# 1:	_____/ 8
# 2:	_____/ 6
# 3:	_____/ 5
# 4:	_____/ 3
# 5:	_____/15
# 6:	_____/ 6
# 7:	_____/ 5
# 8:	_____/ 7
# 9:	_____/ 5
# 10:	_____/11
TOTAL:	_____/71

Good Luck!

OVER. . .

Question 1. [8 MARKS]

In the box beside each piece of code, write its output. If it would generate an error, say so, and give the reason for the error.

Part (a) [1 MARK]

```
L = [5, 6, 7, 8]
for item in L:
    item  = item + 1
print(L)
```

Part (b) [1 MARK]

```
L = [17, 18, 19, 20]
for item in L:
    L[item]  = L[item] + 1
print(L)
```

Part (c) [1 MARK]

```
L = [7, 6, 5, 4]
for item in range(len(L)):
    L[item] = L[item] + 1
print(L)
```

Part (d) [1 MARK]

```
L = [2, 4, 6, 8]
for item in range(len(L)):
    item = item + 1
print(L)
```

Part (e) [1 MARK]

```
L1 = ["I", "like", "running"]
L2 = L1.append("sometimes")
print(L1)
print(L2)
```

Part (f) [1 MARK]

```
n1 = 45
n2 = n1
n2 = n1 + 1
print(n1)
print(n2)
```

Part (g) [1 MARK]

```
def g(d):
    d["Jan"] = "cold!"
months = {"May": "warm"}
g(months)
print(months)
```

Part (h) [1 MARK]

```
def f(s):
    s = s + ' ' + 'sub'
    return s
food = 'sub'
f(food)
print(food)
print(f('tuna'))
```

Question 2. [6 MARKS]

Consider the following function `puzzle` where the type contract is missing.

```
def puzzle(thing):
    for i in range(len(thing)):
        if thing[i - 1] < thing[i]:
            return i - 1
    return -1
```

- Will the function work if `thing` is of type `str`? Circle one:

  YES NO

  If you answered yes, give a value for a string `s` such that `len(s) >= 3`, and the result of calling `puzzle(s)`. If you answered no, explain why calling `puzzle` with a string argument would give an error.

- Will the function work if `thing` is of type `int`? Circle one:

  YES NO

  If you answered yes, give a value for an int `x` such that `x >= 3`, and the result of calling `puzzle(x)`. If you answered no, explain why calling `puzzle` with an integer argument would give an error.

- Will the function work if `thing` is of type `list`? Circle one:

  YES NO

  If you answered yes, give a value for a list `L` such that `len(L) >= 3`, and the result of calling `puzzle(L)`. If you answered no, explain why calling `puzzle` with a list argument would give an error.

Question 3. [5 MARKS]

Complete the header, description, and example calls for the function below.

```
def mystery( [          ] , [          ] , [          ] ) -> [          ]:
    """
    [                                                    ]
    [                                                    ]
    [                                                    ]
    >>> mystery('banana', 'a', 3)
    [                                                    ]
    >>> mystery('hello', 'l', 1)
    [                                                    ]

    """

    count = 0
    result = ""

    for char in s:
        if char == c and count < n:
            result = result + 'X'
            count = count + 1
        else:
            result = result + char

    return result
```

Question 4. [3 MARKS]

Consider the following function header and docstring:

```
def max_divisible_by_three(L: List[int]) -> int:
    """Return the maximum value in L that is divisible by 3.
    If no such value can be found, return -1.

    Precondition: all numbers in L are >= 0
    """
```

Suppose we want to test `max_divisible_by_three`. We have given you one test case as an example. Add three more distinct test cases to the table below that should be used to test this function. No marks will be given for duplicated test cases.

Value of `L`	**Return value**	**Description of Test Case**
`[]`	`-1`	Empty list

Question 5. [15 MARKS]

In the fictional country of Canadia, residents pay income tax on their yearly income.

Each person pays a certain tax percentage (or tax 'rate') depending on what range their income falls in, as in the below table. These income ranges are called *tax brackets*.

Income Range		Tax Rate
	Below $10,000	0%
At least $10,000	Below $20,000	10%
At least $20,000	Below $50,000	20%
At least $50,000	Below $100,000	50%

We summarize these values using the constants `TAX_BRACKET` and `TAX_RATE`, which are parallel lists. They contain the upper ranges and tax rates of each tax bracket.

```
TAX_BRACKET = [10000, 20000, 50000, 100000]
TAX_RATE    = [  0.0,   0.1,   0.2,    0.5]
```

The Canadia government allows for some tax deductions. A person under 25 has their tax rate cut in half. A married person 25 or over has their income reduced by $1,000 prior to calculating their tax rate.

You will write a program that calculates the income tax of Canadia residents listed in a text file.

Part (a) [5 MARKS]

Complete the function `get_tax_rate` according to the docstring below. Only add code in the boxes to make the function work as required.

```
def get_tax_rate(income: int) -> float:
    """Return the tax rate for an individual that makes income (in dollars),
    based on the tax brackets and tax rates in TAX_BRACKET and TAX_RATE.

    If the tax rate for income cannot be found in TAX_BRACKET, return -1.0

    >>> get_tax_rate(10000)
    0.1
    >>> get_tax_rate(250000)
    -1.0
    """

    i = 0
    while [                    ] and [                    ]:
        i = i + 1

    if [                    ]:
        return [                    ]
    return [                    ]
```

We store each person's income information in a plain text file. There are four lines in the file for each person, specifying their name, marital status, age, and income. Each person's set of lines is separated by a line containing `***` (but this line does not appear at the beginning or end of the file). The contents of a file named `tax_info.txt` is shown below.

```
Name: Larradia
single
19
25000
***
Name: Danidia
married
23
30000
***
Name: Alex
married
46
75000
```

Part (b) [5 MARKS]

Complete the function `get_person_list` that will read a file like `tax_info.txt`, and return a list of tuple representations `(is_married, age, income)` of each person. For example, the person named Larradia would be represented as `(False, 19, 25000)`.

Only add code in the boxes to make the function work as required. Note that the tuples in the returned list contain **booleans** and **integers**, and not strings which appear from the file.

```
def get_person_list(f: TextIO) -> List[Tuple]:
    """Return a list of tax information on the persons in a nonempty tax_file.
    Each person's tax information is represented as a tuple (is_married, age, income).

    >>> tax_file = open("tax_info.txt")
    >>> get_person_list(tax_file)
    [(False, 19, 25000), (True, 23, 30000), (True, 46, 75000)]
    """
    person_list = [] # empty list
    next = '***'

    while [                                    ]:
        f.readline()
        is_married = [                                    ]
        age = [                                    ]
        income = [                                    ]

        person_list.append((is_married, age, income))
        next = f.readline().strip()
    return person_list
```

Part (c) [5 MARKS]

Using the functions `get_tax_rate` and `get_person_list` from parts (a) and (b), complete the function `get_income_tax`, which returns a list of the amount of tax the individuals in the tax file must pay. The income tax a person pays is their income multiplied by their tax rate.

```
def get_income_tax_list(tax_file_name: str) -> List[float]:
    """Return a list of the payable income tax for the persons in tax_file_name.

    Recall that persons under 25 years of age have tax rates cut in half, married
    persons 25 and over have their income reduced by $1,000 for computing income tax.
    The $1,000 reduction applies *before* computing the tax rate.

    >>> get_income_tax_list('tax_info.txt')
    [2500.0, 3000.0, 37000.0]
    """
    # Get the tax information using a helper function
    person_list = get_person_list(open(tax_file_name))

    # Initialize the list to return
    tax_list = [] # empty list

    # Write the rest of the function below









    return tax_list
```

Question 6. [6 MARKS]

Answer the questions below about the following function:

```
def print_func(L: List[int]) -> None:
    for i in range(len(L)):
        print(L[i])
        if L[i] > 4:
            for j in range(len(L)):
                print(L[j])
    print("done")
```

Let k be the number of elements in `L`.

Part (a) [1 MARK] Give a formula in terms of k to describe **exactly** how many print statements are executed in the *best case*.

Part (b) [1 MARK] Briefly describe the property of `L` that causes the best case for `print_func`.

Part (c) [1 MARK] Circle the term that best describes the *best case* running time of `print_func`.

constant linear quadratic something else

Part (d) [1 MARK] Give a formula in terms of k to describe **exactly** how many print statements are executed in the *worst case*.

Part (e) [1 MARK] Give an example value for L that produces the worst case running time for `print_func`. Your answer should be a three-element list.

Part (f) [1 MARK] Circle the term that best describes the *worst case* running time of `print_func`.

constant linear quadratic something else

Question 7. [5 MARKS]

A date-to-event dictionary is a dictionary in which the keys are dates (as strings in the format `'YYYY-MM-DD'`), and the values are lists of events that take place on that day.

Complete the following function according to its docstring. **Your code must not mutate the parameter!**

```
def group_by_year(events: Dict[str, List[str]]) -> Dict[str, Dict[str, List[str]]]:
    """Return a dictionary in which in the keys are the years that appear in
    the dates in events, and the values are the date-to-event dictionaries
    for all dates in that year.

    >>> date_to_event = {'2018-01-17': ['meeting', 'lunch'],
                         '2018-03-15': ['lecture'],
                         '2017-05-04': ['gym', 'dinner']}
    >>> group_by_year(date_to_event)
    {'2018': {'2018-01-17': ['meeting', 'lunch'],
              '2018-03-15': ['lecture']},
     '2017': {'2017-05-04': ['gym', 'dinner']}
    }
    """
```

Question 8. [7 MARKS]

In Assignment 3 we built a movie recommender. Read the docstring of `most_popular_in_genre` below carefully, then read the comments outlining the algorithm used to solve the problem. Complete the first part of this function by filling in each box with the letter that corresponds to the line of code that correctly implements that step of the algorithm, so that the function is correct according to its docstring.

```
def most_popular_in_genre(g: str,
                          m: MoviesDict,
                          u: UserRatingDict,
                          n: int) -> List[int]:
    """Return a list containing the ids of movies in m that
    belonging to genre g, sorted by the number of users in
    u that rated each movie, and limited to n movies.

    Recall:
    * MoviesDict = Dict[int, Tuple]
      dictionary of movie id to tuples (movie_name, genres)
    * UserRatingDict = Dict[int, Dict[int, float]]
      dictionary with user ids as keys, and a dictionary of
      movie ids to ratings made by the user as values.

    Precondition: all movie ids in the dictionary u are in m

    >>> movs = {1: ('Titanic', []),
                2: ('Thor', ['Action']),
                3: ('Superman', ['Action'])}
    >>> ratings = {5: {1: 3.0, 2: 3.5, 3: 5.0}, 6: {3: 4.5}}
    >>> most_popular_in_genre('Action', movs, ratings, 1)
    [3]
    >>> most_popular_in_genre('Action', movs, ratings, 2)
    [3, 2]
    """

    # create empty dict to accumulate popularity [ ]

    # iterate over the users [ ]

       # iterate over the movies the user rated [ ]

          # check if the movie is in the correct genre [ ]

             # check if movie is not in the accumulator [ ]

                # start an empty counter for movie [ ]

             # increment movie popularity [ ]

    # sort the movies in acc, get top n (try this after the exam!)
    #...rest of function...
```

(A) `acc = {}`
(B) `acc = 0`
(C) `for uid in u`
(D) `for uid in u[mid]`
(E) `for uid in m`
(F) `for mid in u[uid]`
(G) `for mid in m`
(H) `for mid in m[1]`
(I) `if movs[mid] in g`
(J) `if g in movs[mid]`
(K) `if g in movs[mid][1]`
(L) `if g == movs[mid][1]`
(M) `if mid not in movs`
(N) `if mid not in acc`
(O) `if mid not in acc[1]`
(P) `acc = {mid: 0}`
(Q) `acc[mid] = {}`
(R) `counter = 0`
(S) `counter[mid] = 0`
(T) `acc[mid] = 0`
(U) `acc[counter] = 0`
(V) `mid = 0`
(W) `acc = acc + 1`
(X) `counter = counter + 1`
(Y) `acc[mid] = acc[mid]+1`
(Z) `mid = mid + 1`

Question 9. [5 MARKS]

The following lists have just undergone their first pass in a sorting algorithm. For each list, **circle** which possible sorting algorithms are being used on this list (there could be more than one).

Part (a) [1 MARK]

Original list: `[4, 6, 7, 8, 3, 2]`
First pass: `[4, 6, 7, 8, 3, 2]`

selection sort bubble sort insertion sort

Part (b) [1 MARK]

Original list: `[7, 3, 4, 2, 1, 12]`
First pass: `[3, 4, 2, 1, 7, 12]`

selection sort bubble sort insertion sort

Part (c) [1 MARK]

Original list: `[3, 5, 6, 8, 9]`
First pass: `[3, 5, 6, 8, 9]`

selection sort bubble sort insertion sort

Part (d) [1 MARK]

Original list: `[5, 7, 2, 7, 6]`
First pass: `[2, 7, 5, 7, 6]`

selection sort bubble sort insertion sort

Part (e) [1 MARK]

Original list: `[2, 1]`
First pass: `[1, 2]`

selection sort bubble sort insertion sort

Question 10. [11 MARKS]

In this question, you will develop two classes to represent products in an online shopping cart.

Here is the header and docstring for class Product.

```
class Product:
    """ Information about a product. """
```

Part (a) [2 MARKS]

Here is the header and docstring for method `__init__` in class `Product`. Complete the body of this method

```
    def __init__(self, name: str, price: float, weight: int) -> None:
        """Initialize a new product with product name, price in dollars, and
        weight in grams.

        >>> prod = Product('Science Umbrella', 23.97, 408)
        >>> prod.product_name
        'Science Umbrella'
        >>> prod.price
        23.97
        >>> prod.weight
        408
        """
```

Part (b) [2 MARKS] Complete the body of the `__str__` method in the `Product` class.

```
    def __str__(self) -> str:
        """Return a string representation of this product.

        >>> prod = Product('Science Umbrella', 23.97, 408)
        >>> str(prod)
        'Science Umbrella ($23.97)'
        """
```

Here is the header and docstring for class `ShoppingCart`. In the following questions, you will complete the methods in the class `ShoppingCart`, using the class `Product` in your answers.

```
class ShoppingCart:
    """ Information about a ShoppingCart, which contains zero or more products. """
```

Part (c) [1 MARK] Complete the body of the class `ShoppingCart` method according to its docstring.

```
    def __init__(self, name: str) -> None:
        """Initialize a new shopping cart that is named name with an empty list
        of products.

        >>> cart = ShoppingCart('School Supplies')
        >>> cart.name
        'School Supplies'
        >>> cart.products
        []
        """
```

Part (d) [1 MARK] Complete the body of this class `ShoppingCart` method according to its docstring.

```
    def add_product(self, product: "Product") -> None:
        """Adds the product to the list of products in the ShoppingCart.

        >>> cart = ShoppingCart('School Supplies')
        >>> cart.products
        []
        >>> cart.add_product(Product('Science Umbrella', 23.97, 408))
        >>> len(cart.products)
        1
        """
```

Part (e) [5 MARKS]

Complete the body of this class `ShoppingCart` method according to its docstring.

```
def shipping_cost(self) -> float:
    """Return the cost of shipping the products in the ShoppingCart. Shipping
    is free ($0) if total price of products in ShoppingCart is above $50,
    otherwise the cost depends on total product weight as follows:
        * under 100g            : $ 5.50
        * over 100g, under 500g : $15.00
        * over 500g             : $25.00

    >>> cart = ShoppingCart('School Supplies')
    >>> cart.add_product(Product('Pencil', 3.95, 65))
    >>> cart.shipping_cost()
    5.5
    >>> cart.add_product(Product('Science Umbrella', 23.97, 408))
    >>> cart.shipping_cost()
    15.0
    >>> cart.add_product(Product('Computer Science Umbrella', 24.97, 408))
    >>> cart.shipping_cost()
    0.0
    """
```

DO NOT DETACH THIS PAGE

Use the space on this "blank" page for scratch work, or for any answer that did not fit elsewhere.
Clearly label each such answer with the appropriate question and part number, and refer to this answer on the original question page.

DO NOT DETACH THIS PAGE

Use the space on this "blank" page for scratch work, or for any answer that did not fit elsewhere.
Clearly label each such answer with the appropriate question and part number, and refer to this answer on the original question page.

DO NOT DETACH THIS PAGE
Short Python function/method descriptions:

```
__builtins__:
  input([prompt: str]) -> str
    Read a string from standard input. The trailing newline is stripped. The prompt string,
    if given, is printed without a trailing newline before reading.
  abs(x: float) -> float
    Return the absolute value of x.
  chr(i: str) -> Unicode character
    Return a Unicode string of one character with ordinal i; 0 <= i <= 0x10ffff.
  float(x: object) -> float
    Convert x to a floating point number, if possible.
  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 the list, tuple, dict, or string x.
  max(iterable: object) -> object
  max(a, b, c, ...) -> object
    With a single iterable argument, return its largest item.
    With two or more arguments, return the largest argument.
  min(iterable: object) -> object
  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).
  ord(c: str) -> int
    Return the integer ordinal of a one-character string.
  print(value: object, ..., sep=' ', end='\n') -> None
    Prints the values. Optional keyword arguments:
    sep:  string inserted between values, default a space.
    end:  string appended after the last value, default a newline.
  range([start: int], stop: int, [step: int]) -> list-like-object of int
    Return the integers starting with start and ending with stop - 1 (positive step)
    or stop + 1 (negative step), with step specifying the amount to increment (or decrement).
    If start is not specified, the list starts at 0.  If step is not specified,
    the values are incremented by 1.

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

DO NOT DETACH THIS PAGE

```
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 file F and return the number of characters written.

list:
  x in L --> bool
    Produce True if x is in L and False otherwise.
  L.append(x: object) -> None
    Append x to the end of the list L.
  L.extend(iterable: object) -> None
    Extend list L by appending elements from the iterable. Strings and lists are
    iterables whose elements are characters and list items respectively.
  L.index(value: object) -> int
    Return the lowest index of value in L, but raises an exception if value does
    not occur in S.
  L.insert(index: int, x: object) -> None
    Insert x at position index.
  L.pop([index: int]) -> object
    Remove and return item at index (default last).
  L.remove(value: object) -> None
    Remove the first occurrence of value from L.
  L.reverse() -> None
    Reverse the list *IN PLACE*.
  L.sort() -> None
    Sort the list in ascending order *IN PLACE*.

str:
  x in s --> bool
    Produce True if x is in s and False otherwise.
  str(x: object) -> str
    Convert an object into its string representation, if possible.
  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.endswith(S2: str) -> bool
    Return True if and only if S ends with S2.
  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.index(sub: str) -> int
    Like find but raises an exception if sub does not occur in S.
```

DO NOT DETACH THIS PAGE

```
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.isdigit() -> bool
  Return True if all characters in S are digits
  and there is at least one character in S, and False otherwise.
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.lstrip([chars: str]) -> str
  Return a copy of the string S with leading whitespace removed.
  If chars is given and not None, remove characters in chars instead.
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.rstrip([chars: str]) -> str
  Return a copy of the string S with trailing whitespace removed.
  If chars is given and not None, remove characters in chars instead.
S.split([sep: str]) -> List[str]
  Return a list of the words in S, using string sep as the separator and
  any whitespace string if sep is not specified.
S.startswith(S2: str) -> bool
  Return True if and only if S starts with S2.
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.
```