CSC 148H5 S 2017 Test 1
Duration — 50 minutes
Aids allowed: none

Student Number: |_|_|_|_|_|_|_|_|_|_|_|_|

Last Name: _____ **First Name:** _____

☐ Lecture Section: L0103 Instructor: Dan Zingaro (13:00-14:00)
☐ Lecture Section: L0101 Instructor: Samar Sabie (9:00-10:00)
☐ Lecture Section: L0102 Instructor: Ritu Chaturvedi (10:00-11:00)

---

*Do **not** turn this page until you have received the signal to start.*
(Please fill out the identification section above, **write your name on the back of the test**, and read the instructions below.)
Good Luck!

This test consists of 4 questions on 10 pages (including this page). *When you receive the signal to start, please make sure that your copy is complete.* Comments are not required except where indicated, although they may help us mark your answers. They may also get you part marks if you can't figure out how to write the code.
If you use any space for rough work, indicate clearly what you want marked.

1: _____/ 6
2: _____/ 6
3: _____/ 5
4: _____/ 5

TOTAL: _____/22

Question 1. [6 MARKS]

Complete the `push` method for class `SpecialStack`. `SpecialStack` is similar to the `Stack` class that you saw in lecture, but it has a maximum capacity and can only hold data of type `int`. If the `SpecialStack` is already at full capacity, raise a `StackFullException` with the error message "Sorry. Maximum capacity reached". Otherwise, if the item to be pushed is not an `int`, raise a `TypeError` with the message "Sorry. Only integers are permitted". Make sure to define all necessary exceptions (if any). You can implement `SpecialStack` with the top of stack at the right or the left. Do not implement any extra methods.

```
class SpecialStack:

    '''A last-in, first-out (LIFO) stack of integer items and limited capacity'''

    def __init__(self, max_capacity):
        '''(SpecialStack, int) -> None
        A new empty SpecialStack.'''
        self._data = []
        self._max_capacity = max_capacity


    def push(self, item):
        '''(SpecialStack, int) -> None
        Place item on top of the stack.

        >>> s = SpecialStack(2)
        >>> s.push(1)
        >>> s.push('a')
        ... TypeError raised: Sorry. Only integers are permitted
        >>> s.push(2)
        >>> s.push(3)
        ... StackFullException raised: Sorry. Maximum capacity reached
        '''
```

Question 2. [6 MARKS]

Explain in plain English the purpose of the following `mystery` function. (Remember: this means that we want the overall purpose of the code, not a line-by-line description of what the code does.). Make sure to state **parameter types**. You can assume that all necessary modules have been imported and that o1 and o2 are not empty.

```
def mystery(o1, o2):

    x = []
    y = []

    while not o1.is_empty():
        a = o1.pop()
        x.append(a)

    while not o2.empty():
        b = o2.dequeue()
        y.append(b)

    o1.push(b)
    x.reverse()
    for i in range(1, len(x)):
        o1.push(x[i])

    for j in range(0, len(y)-1):
        o2.enqueue(y[j])
    o2.enqueue(a)
```

Question 3. [5 MARKS]

Consider the following two classes:

```
class Car:
    '''A Car object'''

    def __init__(self, model, year):
        '''(Car, str, int) -> None
        Create a Car with the given model and year.'''
        self.model = model
        self.year = year
        print("Car object created")

    def get_CO2_emissions(self):
        '''(Car) -> int
        Return the number of grams of carbon dioxide (CO2) that are produced
        from driving Car for one kilometer.'''
        return 300

    def change_oil(self):
        '''(Car) -> str
        Change the Car oil'''
        return "Oil changed"

    def __repr__(self):
        '''(Car) -> str'''
        return "A " + self.model + " " + str(self.year) + " car"



class HybridCar(Car):
    '''A Hybrid Car Object.'''

    def __init__(self, model, year, engine_type):
        '''(HybridCar, str, int, str) -> None
        Create a HybridCar with the given model, year and engine_type.'''
        Car.__init__(self, model, year)
        self.engine_type = engine_type
        print("Hybrid Car object Created")

    def get_CO2_emissions(self):
        '''(HybridCar) -> int
        Return the number of grams of carbon dioxide (CO2) that are produced
        from driving HybridCar for one kilometer.'''
        return 150
```

```
    def change_breaks(self):
        '''(HybridCar) -> str
        Change the breaks.'''
        return "Breaks changed"

    def print(self):
        return "A " + self.model + " " + str(self.year) + " hybrid car with a "\
               + self.engine_type + " engine"
```

Write underneath each statement the expected output or error message:

```
>>> c1 = Car("Toyota Camry", 1995)



>>> c2 = HybridCar("Toyota Prius", 2017, "electric-petroleum")



>>> c1.change_breaks()



>>> c1.get_CO2_emissions()



>>> c2.change_oil()



>>> c2.get_CO2_emissions()



>>> print(c2)
```

Question 4. [5 MARKS]

Write the below function that returns the number of digits in integer `n`. **Your code must be recursive** and you must **not** use any string methods. Your code does not need to handle wrong input types such as floats.

```
def number_of_digits(n):
    '''(int) -> int
    Return the number of digits in n. n >= 0.

    >>> number_of_digits(2)
    1
    >>> number_of_digits(8272)
    4
    '''
```

[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.]

[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.]

Last Name: _____ **First Name:** _____

Short Python function/method descriptions:

```
__builtins__:
  input([prompt]) -> str
    Read a string from standard input; return that string with no newline.
    The prompt string, if given, is printed without a trailing newline before reading.
  isinstance(object, class) -> bool
    Return whether an object is an instance of a class or of a subclass thereof.
  print(value, ..., sep=' ', end='\n') -> NoneType
    Print the values. Optional keyword arguments:
    sep:  string inserted between values, default a space.
      end:  string appended after the last value, default a newline.
int:
  int(x) -> int
    Convert a string or number to an integer, if possible.
    A floating point argument will be truncated towards zero.
list:
  L.append(object) -> None -- Append object to end.
  L.insert(index, object) -> None -- Insert object before index.
str:
  S.count(sub[, start[, end]]) -> 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[,i]) -> 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.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 S converted to lowercase.
  S.replace(old, new) -> str
    Return a copy of string S with all occurrences of the string old replaced with the string new.
  S.split([sep]) -> list of 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(prefix) -> bool
    Return True if S starts with the specified prefix and False otherwise.
  S.strip() -> str
    Return a copy of S with leading and trailing whitespace removed.
  S.upper() -> str
    Return a copy of S converted to uppercase.
```