Last Name: _____ First: _____ Netid: _____

CS 1110 Prelim 2 November 11th, 2021

Orange text = comments by Spring 2022 instructors

This 90-minute exam has 5 questions worth a total of 100 points. Scan the whole test before starting. Budget your time wisely. Use the back of the pages if you need more space. You may tear the pages apart; we have a stapler at the front of the room.

It is a violation of the Academic Integrity Code to look at any exam other than your own, look at any reference material, or otherwise give or receive unauthorized help.

You will be expected to write Python code on this exam. We recommend that you draw vertical lines to make your indentation clear, as follows:

```
def foo():
    if something:
        do something
        do more things
    do something last
```

You should not use while-loops on this exam. Beyond that, you may use any Python feature that you have learned about in class (if-statements, try-except, lists, for-loops, recursion and so on).

Question	Points	Score
1	2	
2	20	
3	26	
4	26	
5	26	
Total:	100	

The Important First Question:

1. [2 points] Write your last name, first name, and netid, at the top of *each* page.

Last Name: _____ First: _____ Netid: _____

Reference Sheet

String Operations

Operation	Description
`len(s)`	**Returns**: Number of characters in `s`; it can be 0.
`a in s`	**Returns**: True if the substring `a` is in `s`; False otherwise.
`a*n`	**Returns**: The concatenation of `n` copies of `a`: `a`+`a`+$\cdots$+`a`.
`s.find(s1)`	**Returns**: Index of FIRST occurrence of `s1` in `s` (-1 if `s1` is not in s).
`s.count(s1)`	**Returns**: Number of (non-overlapping) occurrences of `s1` in `s`.
`s.islower()`	**Returns**: True if `s` is *has at least one letter* and all letters are lower case; it returns False otherwise (e.g. `'a123'` is True but `'123'` is False).
`s.isupper()`	**Returns**: True if `s` is *has at least one letter* and all letters are uppper case; it returns False otherwise (e.g. `'A123'` is True but `'123'` is False).
`s.isalpha()`	**Returns**: True if `s` is *not empty* and its elements are all letters; it returns False otherwise.
`s.isdigit()`	**Returns**: True if `s` is *not empty* and its elements are all numbers; it returns False otherwise.
`s.isalnum()`	**Returns**: True if `s` is *not empty* and its elements are all letters or numbers; it returns False otherwise.

List Operations

Operation	Description
`len(x)`	**Returns**: Number of elements in list `x`; it can be 0.
`y in x`	**Returns**: True if `y` is in list `x`; False otherwise.
`x.index(y)`	**Returns**: Index of FIRST occurrence of `y` in `x` (error if `y` is not in `x`).
`x.count(y)`	**Returns**: the number of times `y` appears in list `x`.
`x.append(y)`	Adds `y` to the end of list `x`.
`x.insert(i,y)`	Inserts `y` at position `i` in `x`. Elements after `i` are shifted to the right.
`x.remove(y)`	Removes first item from the list equal to `y`. (error if `y` is not in `x`).

Dictionary Operations

Function or Method	Description
`len(d)`	**Returns**: number of keys in dictionary `d`; it can be 0.
`y in d`	**Returns**: True if `y` is a key `d`; False otherwise.
`d[k] = v`	Assigns value `v` to the key `k` in `d`.
`del d[k]`	Deletes the key `k` (and its value) from the dictionary `d`.
`d.clear()`	Removes all keys (and values) from the dictionary `d`.

Last Name: _____ First: _____ Netid: _____

2. [20 points total] **Iteration**.

   Implement the functions below according to their specification using for-loops. You **do not** need to enforce preconditions. But pay attention to all instructions.

   (a) [8 points] Alternate version of first line of specification: "Returns a copy of d but where each key in the new dictionary is 1 + the corresponding key in d.". Also, assume d is not empty.

```
def shiftkeys(d):
    """Returns a copy of d with 1 added to each key
    Example: shiftkeys({1:'a',2:'b'}) returns {2:'a',3:'b'}.
    Precondition: d is a dictionary whose keys are ints"""

    result = {}   # Accumulator

    # Looping over dictionary loops over keys
    for k in d:
        result[k+1] = d[k]

    return result
```

   (b) [12 points] For the purposes of this question (only), define the "average" of an empty list of numbers to be 0.0.
   For this question, you may not use Python's prebuilt sum() or numpy's mean() function.

```
def collapse(ragged):
    """MODIFIES the 2d list ragged, converting each element list into its average
    As a result, the 2d list will become a 1d list of floats. Note that the
    elements of ragged can be lists of different sizes or even empty.
    Example: If ragged is [[1.0,2.0],[3.2],[]], then collapse(ragged) converts the
    list ragged into the list [1.5, 3.2, 0.0]
    This is a procedure and should not return a value.
    Precondition: ragged is a nonempty list whose elements are (possibly empty)
    lists of floats"""

    # To modify the list we have to loop over range
    for pos in range(len(ragged)):
        # Intermediate accumulator
        accum = 0.0
        for item in ragged[pos]:
            accum = accum + item

        # Compute the average
        if len(ragged[pos]) > 0:
            accum = accum/len(ragged[pos])

        # Reassign the element
        table[pos] = accum
```

Alternate (not-so-different) solution:

```
for i in range(len(ragged)):
    row = ragged[i]
    if len(row) == 0:
        ragged[i] = 0.0
    else:
        sum_so_far = 0.0
        for num in row:
            sum_so_far+= num
        ragged[i] = sum_so_far/len(row)
```

Last Name: _____ First: _____ Netid: _____

3. [26 points total] **Recursion**.

   Use recursion to implement the following functions. Solutions using loops will receive no credit. You **do not** need to enforce the preconditions. But pay attention to all instructions. When writing recursive code in real life (not on exams), assertions regarding the types of arguments are invaluable.

   (a) [16 points] A different way to state the 2nd line of the specification: "The list returned is a new list, even if its contents happen to be the same as one of its input lists."
   This function has a nice recursive solution. Although loops were disallowed for this function, we note that there is a natural while-loop solution, although there does not seem to be a natural for-loop solution.

```
def merge(a,b):
    """Returns the sorted merge of the sorted lists a, b.
    The list returned is a copy of a and/or b.
    Examples: merge([],[]) returns [] (a COPY)
              merge([],[1,2]) returns [1,2] (a COPY)
              merge([0,3],[1,3,4]) returns [0,1,3,3,4]
    Precondition: a and b are (possibly empty) sorted lists of ints"""
    # HINT: You should divide-and-conquer on either a or b but NOT
    # both at the same time. You may NOT use sort() or similar methods/functions.

    # Base case is when either is empty
    if len(a) == 0:
        return b[:]
    elif len(b) == 0:
        return a[:]

    # Which one to divide depends on order of first element
    first = a[0]
    secnd = b[0]
    if first < secnd:
        # a has lowest element.  Pull it off and recurse
        left  = [first]
        right = merge(a[1:],b)
    else:
        # b has lowest element.  Pull it off and recurse
        left  = [secnd]
        right = merge(a,b[1:])

    # Normal combination step
    return left+right
```

Alternate solution:

```
if len(a) == 0 or len(b) == 0:
     return a+b  # This is a new list

# If here, both lists are non-empty
if a[0] <= b[0]:
     return [a[0]]+merge_recursive(a[1:], b)
else:
     return [b[0]]+merge_recursive(a,b[1:])
```

(b) [10 points] (Solutions using loops will receive no credit) Although loop-based solutions were banned for this question, a for-loop is really the natural way to solve this problem.

```
def toevens(nums):
    """Returns a copy of nums where every odd number is increased by 1.
    Example: toevens([0,1,2,3,4]) returns [0,2,2,4,4] (evens unaffected)
    Precondition: nums is a (potentially empty) list of ints"""

    if len(nums) == 0:
        return []
    elif len(nums) == 1:
        # Is the number even or odd?
        if nums[0] % 2 == 0:
            return [nums[0]]
        else:
            return [nums[0]+1]

    left  = toevens(nums[:1])
    right = toevens(nums[1:])

    return left+right
```

Alternate solution. Uses a helper, and a trick that if a list x has length 1, x[1:] evaluates to the empty list (it does NOT yield an error in Python).

```
    if len(nums) == 0:
        return []

    # If here, nums[0] exists
    front = nums[0]
    if is_odd(front):
        front = front + 1
    return [front] + toevens(nums[1:])

def is_odd(num):
    """Returns True if num is odd, False o.w.
    Precondition: num is an int"""
    if num % 2 == 1:
        return True
    else:
        return False
```

4. [26 points total] **Classes and Subclasses**

The Exerpy system that you use to complete your labs is completely written in Python (using a module/package known as Django). This was a natural choice since your lab activities are in Python. But we actually chose it because it makes it easy for us to quickly add new question types. Exerpy has support for a broad range of questions, including multiple choice, free response, code snippets, and drag-and-drop vocabulary matching.

To make this possible, Exerpy has a class `Question` which is the parent class for every question type. Then each type of question has its own class. So vocabulary questions are represented by the `Vocabulary` class and code questions are represented by the `Script` class.

In this question, you will implement a simplified version of two classes from the Exerpy system. You will implement the base class `Question` and the subclass `Choice`, which is used for multiple choice questions. The attributes for these classes are as follows:

`Question`

Attribute	**Invariant**	**Category**
`USED_INDICES`	list of all active question indices (ints)	Class attribute
`_index`	an int > 0	**Immutable** instance attribute
`_text`	a nonempty string	**Mutable** instance attribute

`Choice` (in addition to those inherited)

Attribute	**Invariant**	**Category**
`_choices`	nonempty tuple of strings	**Mutable** instance attribute
`_correct`	int and a valid index of `_choices`	**Mutable** instance attribute

Note that a *valid index* can be negative, as `_choices[-1]` is the last element of `_choices`.

Instructions: On the next four pages, you are to do the following:

1. Fill in the missing information in each class header.
2. Add any necessary class attributes
3. Add getters and setters as appropriate for the instance attributes
4. Fill in the parameters of each method (beyond the getters and setters)
5. Implement each method according to the specification
6. Enforce any preconditions in these methods using asserts

We have not added headers for the getters and setters. You are to write these from scratch. However, **you are not expected to write specifications for them**. For the other methods, pay attention to the provided specifications. The only parameters are those in the preconditions.

The class `Choice` **may not** use any attribute or getter/setter inherited from `Question`. It may only use `super()` to access overridden methods. Because we did not cover the material in time for this exam, you should use `type` instead of `isinstance` to check type-based preconditions.

(a) [12 points] The class `Question` It is OK to omit the "(object)" from the header of the class Question.

```
class Question(object):                                    # Fill in missing part
    """A class representing a question in the lab system

    Attribute USED_INDICES: A CLASS ATTRIBUTE list of all question indices.
    This list starts off empty, as there are no questions to start with."""
    # ATTRIBUTE _index: The question index. An int > 0          (IMMUTABLE)
    # ATTRIBUTE _text: The question wording. A nonempty string (MUTABLE)

    # CLASS ATTRIBUTE. NO GETTERS OR SETTERS.
    USED_INDICES = []


    # DEFINE GETTERS/SETTERS/HELPERS AS APPROPRIATE. SPECIFICATIONS NOT NEEDED.
    def getIndex(self):
        """Returns the index number for this question"""
        return self._index

    def getText(self):
        """Returns the wording for this question"""
        return self._text

    def setText(self,value):
        """Sets the wording for this question

        Parameter value: The new wording
        Precondition: value is a nonempty string
        assert type(value) == str
        assert value != ''
        self._text = value
```

```
# Class Question (CONTINUED).

def __init__(self,    index,    text):                        # Fill in missing part
    """Initializes a new question for the given index and text

    No question can share the same index as another. On creation, the
    question index is added to the class attribute USED_INDICES.

    Precondition: index is an int > 0, and not already in use.
    That is, index cannot be an element of USED_INDICES.
    Precondition: text is a non-empty string"""

    assert type(index) == int and index > 0
    assert type(text) == str and text != ''
    assert not index in Question.USED_INDICES
    self._index = index
    self._text = text
    Question.USED_INDICES.append(index)




def __str__(self):                                           # Fill in missing part
    """Returns a string representation of this question.

    The format is '<index>. <text>'.

    Example: If index is 2 and the text is 'What is your quest?',
    this method will return '2. What is your quest?' """

    return str(self._index)+'. '+self._text



def __eq__(self, other):                                     # Fill in missing part
    """Returns True if self and other are equal. False otherwise.

    An object is equal to this one (self) if it has the same type
    and the same index. You do not need to compare text, since
    indices are unique.

    Precondition: NONE. other can be ANYTHING """

    return type(self) == type(other) and self.index == other.index
```

(b) [14 points] The class `Choice`.

```
class Choice(question):                                    # Fill in missing part
    """A class representing a multiple choice question."""
    # ATTRIBUTE _choices: The options. A nonempty tuple of strings.    (MUTABLE)
    # ATTRIBUTE _correct: The index of the correct answer. An int.     (MUTABLE)
    # ADDITIONAL INVARIANT: _correct is a valid index of _choices at all times
    # HINT: This allows _correct to be negative as long as it is in bounds

    # DEFINE GETTERS/SETTERS/HELPERS AS APPROPRIATE. SPECIFICATIONS NOT NEEDED.


    def getChoices(self):
        """Returns the options to choose"""
        return self._choices


    def setChoices(self,value):
        """Sets the options to choose

        Precondition: value is a nonempty tuple of strings.
        getCorrect() is a valid index of value."""
        assert type(value) == tuple
        assert len(value) > 0
        for item in value:
            assert type(item) == str
        assert self._correct < len(value)
        assert self._correct >= -len(value)
        self._choices = value


    def getCorrect(self):
        """Returns the index of the correct answer"""
        return self._correct


    def setCorrect(self,value):
        """Sets the index of the correct answer

        Precondition: value is an int and a valid index of getChoices()."""
        assert type(value) == int
        assert value < len(self._choices)
        assert value >= -len(self._choices)
        self._correct = value
```

```
# Class Choice (CONTINUED).
# REMEMBER: This page may not use any attributes or getters/setters from Question.


def __init__(self, index, text, choices, correct=-1):        # Fill in missing part
    """Initializes a new multiple choice question with given choices.

    Precondition: index is an int > 0, and not already in use.
    That is, index cannot be an element of USED_INDICES in Question.
    Precondition: text is a non-empty string
    Precondition: choices is a nonempty tuple of strings
    Precondition: correct is an int and a valid index of choices
    (OPTIONAL ATTRIBUTE; correct is -1, the last choice, by default)"""

    # User super() to initialize index and text
    super().__init__(index,text)


    # We CANNOT use the setters for correct and choices, as the setters
    # assume both attributes already exist. We need to set one manually.
    assert type(choices) == tuple
    assert type(correct) == int
    assert correct < len(choices)
    assert correct >= -len(choices)

    self._correct = correct
    self.setChoices(choices)    # NOW a setter is okay









def __str__(self):                                           # Fill in missing part
    """Returns a string representation of this multiple choice question.

    The format is '<index>. <text> <answer>'. For example, suppose the
    question with index 2 and text 'What is your quest?' has choices
    ('To pass this exam.', 'To seek the Holy Grail.'). If correct is 1,
    then the string is '2. What is your quest? To seek the Holy Grail.' """

    return super().__str__()+' '+self._choices[self._correct]
```

Last Name: _____ First: _____ Netid: _____

5. [26 points total] **Folders and Name Resolution**

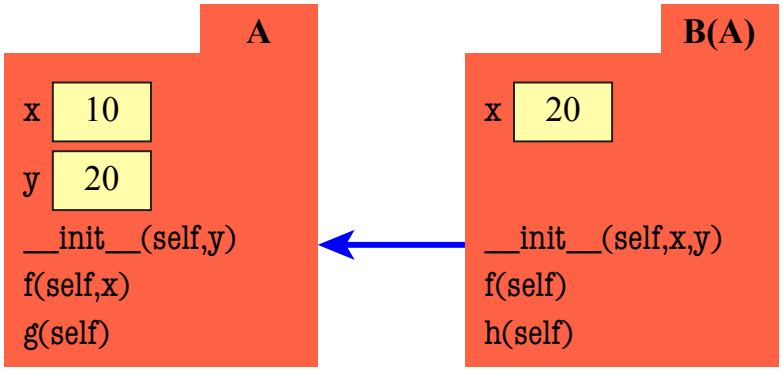
Consider the two (undocumented) classes below, together with their line numbers.

```
1  class A(object):
2      x = 10
3      y = 20
4
5      def __init__(self,y):
6          self.z = y
7          self.x = self.f()
8
9      def f(self,x=5):
10         return x*self.y
11
12     def g(self):
13         return self.x+self.y
```

```
15 class B(A):
16     x = 20
17
18     def __init__(self,x,y):
19         self.y = x
20         super().__init__(x)
21
22     def f(self):
23         return self.x*self.y
24
25     def h(self):
26         y = self.x+self.z
27         return y+self.y
```

(a) [5 points] Draw the class folders in the heap for these two classes.

OK if the signature for f in class A says "f(self,x=5)", but NOT OK to have "f(self)".



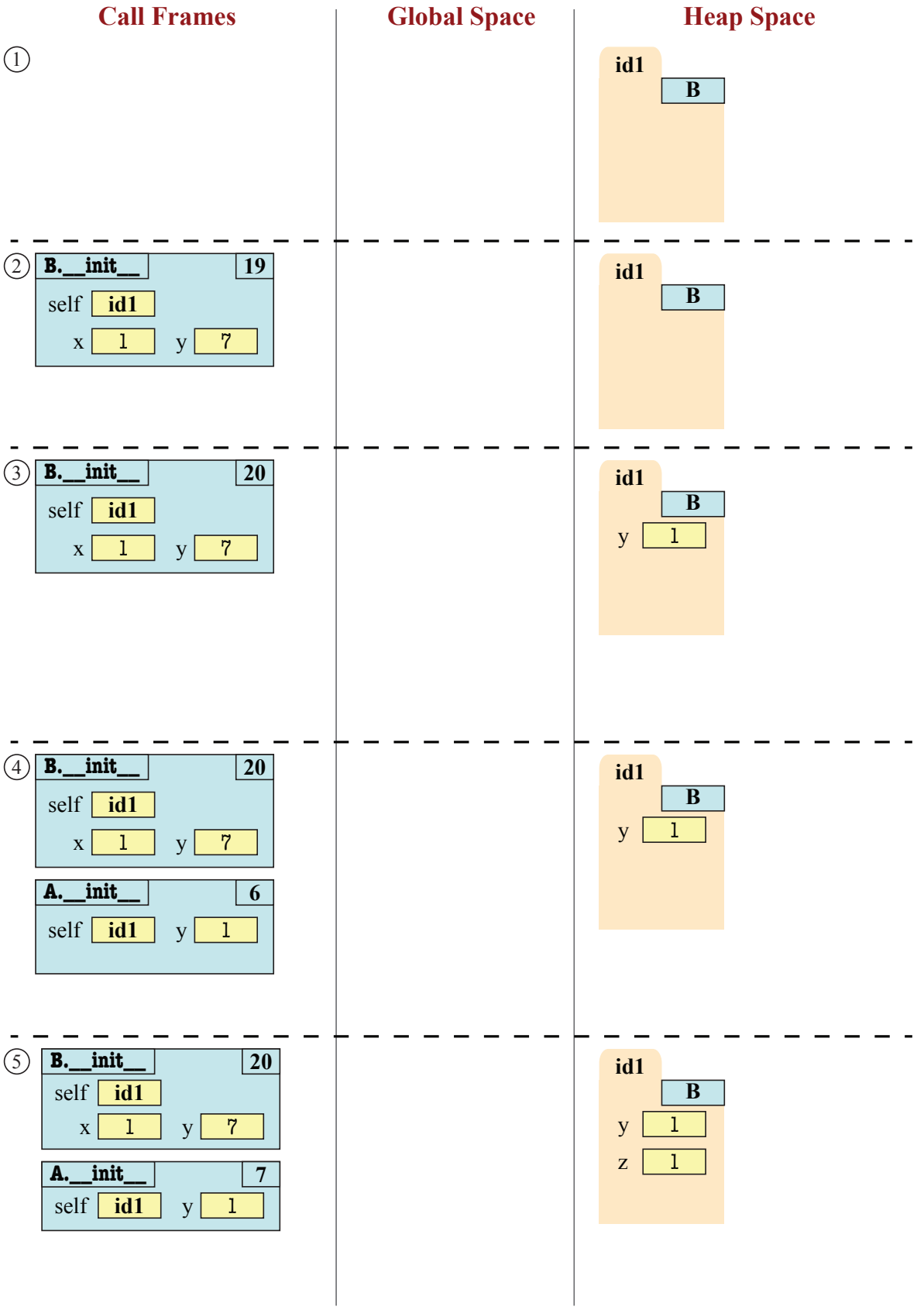
(b) [21 points] On the next two pages, diagram the call

```
>>> b = B(1,7)
```

You will need **ten diagrams**. Draw the call stack, global space and the heap. If the contents of any space are unchanged between diagrams, you may write *unchanged*. You do not need to draw the class folders from part (a).

When diagramming a constructor, you should follow the rules from Assignment 5. Remember that `__init__` is a helper to a constructor but it is not the same as the constructor. In particular, there is an important first step before you create the call frame.

Last Name: _____ First: _____ Netid: _____



Last Name: _____ First: _____ Netid: _____