Last Name (PRINT): _____ **First Name:** _____

Student Number: |_|_|_|_|_|_|_|_|_|_| **UTORid:** |_|_|_|_|_|_|_|_|

Signature: _____

UNIVERSITY OF TORONTO MISSISSAUGA
APRIL 2022 EXAMINATIONS

CSC148H5S
Introduction to Computer Science

Instructor(s): Bogdan Simion, Sonya Allin, Pooja Vashisth

Duration: 3 hours
Examination Aids: Provided aid sheet at the end of this exam

The University of Toronto Mississauga and you, as a student, share a commitment to academic integrity. You are reminded that you may be charged with an academic offence for possessing any unauthorized aids during the writing of an exam. Clear, sealable, plastic bags have been provided for all electronic devices with storage, including but not limited to: cell phones, smart watches, SMART devices, tablets, laptops, and calculators. Please turn off all devices, seal them in the bag provided, and place the bag under your desk for the duration of the examination. You will not be able to touch the bag or its contents until the exam is over.

If, during an exam, any of these items are found on your person or in the area of your desk other than in the clear, sealable, plastic bag, you may be charged with an academic offence. A typical penalty for an academic offence may cause you to fail the course.

*Please note, once this exam has begun, you **CANNOT** re-write it.*

Please read the following instructions carefully.

- Please fill out the information at the top of this cover page.
- This examination has **10** questions. There are a total of **20 pages**, **DOUBLE-SIDED**.
- **DO NOT** open or turn over the exam paper until the exam has started.
- You may always write helper functions unless asked not to.
- Documentation is *not* required unless asked for.
- Answer questions clearly and completely, illegible answers will not be marked. Provide justification unless explicitly asked not to.
- **You must earn a grade of at least 40% on this exam to pass this course**. Otherwise, your final course grade will be set no higher than 47%

Question	Grade	Out of
Q1		8
Q2		4
Q3		7
Q4		6
Q5		3
Q6		6
Q7		8
Q8		6
Q9		6
Q10		6
Total		60

Use this page for rough work. If you want work on this page to be marked, please indicate this clearly *at the location of the original question.*

1. **[8 marks] Memory Model and Tracing Code** Estimated Completion: 15 minutes

(a) **[2 marks]** You are given the code below.

```
1  def perform_some_other_action(L: list) -> None:
2      for elem in L:
3          elem = elem[0]
4          L.append(elem)
5          if len(L) == 0:
6              perform_some_other_action(L)
7
8  lst = [[1], (2, 3), ['blah']]
9  perform_some_other_action(lst)
10 print(lst)
```

Circle the outcome of running this code. **Read the code very carefully!**

A. [1], (2, 3), ['blah']
B. [[1], (2, 3), ['blah'], 1, 2, 'blah']
C. [[1], (2, 3), ['blah'], [1], [1], [1]]
D. TypeError: 'int' object is not subscriptable
E. The program will result in an infinite loop
F. RecursionError: maximum recursion depth exceeded

(b) **[2 marks]** You are given the code below.

```
1  def fancy_function(n: int) -> int:
2      if n == 0:
3          return n
4      else:
5          return 1 + fancy_function(n - 1)
6
7  lst = [11, 16, -3, 22]
8  for index in range(len(lst)):
9      if index % 2 == 0:
10         lst[index] = fancy_function(lst[index])
11     else:
12         lst[index] += 1
13 print(lst)
```

Circle the outcome of running this code. **Read the code very carefully!**

A. [11, 16, -3, 22]
B. [11, 17, -3, 23]
C. [12, 17, -2, 23]
D. [0, 17, 0, 23]
E. [12, 16, -2, 22]
F. RecursionError: maximum recursion depth exceeded

(c) **[4 marks] Tracing Code**

Consider the following code and **read it carefully**:

```
1  class Good:
2      def __init__(self) -> None:
3          self.num = 0
4          self.items = []
5
6      def add(self, val: float, item: str) -> None:
7          if item not in self.items:
8              self.num += val
9              self.items.append(item)
10
11     def remove(self, val: float, item: str) -> None:
12         if item in self.items:
13             self.items.remove(item)
14             self.num -= val
15
16 class Mythical(Good):
17     def __init__(self) -> None:
18         Good.__init__(self)
19         self.inventory = {}
20
21     def add(self, val: float, item: str) -> None:
22         Good.add(self, val, item)
23         if item in self.inventory:
24             self.inventory[item] += val
25         else:
26             self.inventory[item] = val
27
28 class Morning(Good):
29     def __init__(self) -> None:
30         Good.__init__(self)
31         self.inventory = {}
32
33     def remove(self, val: float, item: str) -> None:
34         if item in self.items:
35             self.items.remove(item)
36             self.num -= val
37         self.inventory[item] = self.num
```

For each of the following independent programs, write down the correct output immediately below it. If the program raises an Exception, indicate what line does so and explain why.

PROGRAM 1:

```
b = Mythical()
b.add(7, 'Rhett')
b.add(3, 'Link')
b.add(0, 'Stevie')
b.remove(3, 'Link')
print(b.items, b.num)
print(b.inventory)
```

PROGRAM 2:

```
b, c = Mythical(), Morning()
b.add(0, 'Stevie')
b.add(3, 'Link')
b.add(7, 'Rhett')
c.add(0, 'Stevie')
c.add(3, 'Link')
c.add(7, 'Rhett')
b.remove(3, 'Link')
c.remove(3, 'Link')
if isinstance(b, Good) and isinstance(c, Good):
    if b.num == c.num or b.items == c.items:
        print(b.inventory == c.inventory)
        print(b.inventory, c.inventory)
    else:
        print(b.inventory, c.inventory)
        print(b.inventory == c.inventory)
```

Use this page for rough work. If you want work on this page to be marked, please indicate this clearly *at the location of the original question.*

2. **[4 marks] OOP 1** Estimated Completion: 5 minutes

Fill in the blanks in the code below, to implement the methods indicated:

```
class WeirdList(list):
    """ Class which subclasses Python's list. """

    def __str__(self) -> str:
        """ Return a string containing the size of the list.
        >>> f = WeirdList([1, 2, 3])
        >>> str(f)
        '3'
        """
        # TODO: Fill in the blank to make this method work correctly for any WeirdList!

        return ____________________________

    def last(self) -> Any:
        """ Return the last element in the list.
        Precondition: the list has at least 1 element.

        >>> f = WeirdList([1, 4, 25])
        >>> f.last()
        25
        """
        # TODO: Fill in the blank to make this method work correctly for any WeirdList!

        return ____________________________
```

3. **[7 marks] OOP 2** Estimated Completion: 12 minutes

You are given the following classes:

```
class Account:
    """ An Account class.
    === Representation Invariants ===
    self.rewards >= 0
    """
    rewards: int
    balance: int

    def __init__(self, rewards: int, balance: int) -> None:
        self.rewards = rewards
        self.balance = balance

    def add_balance(self, balance: int) -> None:
        raise NotImplementedError

    def perform_transaction(self, magic_number: int) -> None:
        raise NotImplementedError

class PremiumAccount(Account):
    """ A PremiumAccount class. """
    level: int

    def __init__(self, rewards: int, balance: int) -> None:
        LINE OF CODE OMITTED HERE
        self.level = 0
        self.x = {}

    def add_balance(self, balance: int) -> None:
        self.balance += balance

    def perform_transaction(self, magic_number: int) -> None:
        self.rewards = magic_number * random.randint(0, 100)
        if self.level in self.x:
            self.x[self.level].append((magic_number, "gold badge"))
        else:
            self.x[self.level] = [(magic_number, "silver badge")]
```

(a) **[2 marks]** Replace "LINE OF CODE OMITTED HERE" from the code above, with ONE line of code which makes the initialization of a PremiumAccount instance possess all the attributes it needs (hint: review the entire code provided to determine what it needs). Your answer must be a valid Python statement, and it MUST follow OOP principles you learned in class.

(b) **[1 mark]** Write below the **correct and full** type contract for the attribute x.

(c) **[2 marks]** In the spaces below, indicate the methods which need **explicit preconditions** to ensure that the explicit representation invariant is not violated.

EXTRA REQUIREMENTS for this question (MUST READ CAREFULLY to avoid losing marks!):

- The method names you write for each of the classes below, MUST be different.
- For each precondition, you MUST write a syntactically valid Python statement, not plain English wording.
- Only one method or one precondition must be filled in each of the blanks, as indicated after each colon sign (:).

Class Account's method: _____ - precondition: _____

Class PremiumAccount's method: _____ - precondition: _____

(d) **[2 marks]** You have been chosen to design the following map-related classes: `Continent`, `Country`, and `Canada`. Your tasks are as follows:

i. Identify which Object-Oriented Programming **relationships** are most appropriate among these three classes if you were to you design them. State the relationships explicitly and state how they apply to which classes.

ii. Explain in words how an implementation of these relationships would look like for these classes. Be explicit.

Note: You do not have to write an implementation, but you **must** state the relevant aspects below, in your own words. For example, state a one-line class declaration, if it is relevant. Mention what type of attributes you would need, if any are relevant to illustrate your point.

4. **[6 marks] LinkedLists** Estimated Completion: 15 minutes

Consider the following code implementing a doubly-linked list, which has `_DLLNode` objects as nodes.

A `_DLLNode` object has both a `next` reference to the following node in the list, as well as a `prev` reference to the preceding node in the list.

The list does not wrap around! That is, the last node's `next` attribute references None, and the first node's `prev` references None.

```
class _DLLNode:
    """A node in a linked list.

    === Public Attributes ===
    item: The data stored in this node
    next: The next node in the list, or None if there are no further nodes after this node.
    prev: The previous node in the list, or None if there is no node before this one.
    """
    item: Any
    next: Optional[_DLLNode]
    prev: Optional[_DLLNode]

    def __init__(self, item: Any) -> None:
        """Initialize a new node storing <item>, with no next or prev node."""
        self.item = item
        self.next = None
        self.prev = None

class CustomDLL:
    """ A customized doubly-linked list. """
    _first: Optional[_DLLNode]

    def __init__(self, lst: list[int]) -> None:
        """Initialize this doubly-linked list."""
        # Intentionally omitted the init implementation, for space. Assume it exists and it's correct.
```

Implement the `insert_last` method below. **Read the docstring carefully, including the doctests.**

```
    def insert_last(self, value: Any, after: Any) -> bool:
        """Insert a new Node with the value <value> after the LAST occurrence of the
        value <after> in this list. If <after> does not exist in the list, then do not
        insert anything and return False.

        The list must be correctly linked after this operation.

        >>> sl = CustomDLL([7, 2, 7, 3])
        >>> str(sl)   # You can assume that __str__ is implemented already.
        '7 2 7 3'
        >>> sl.insert_last(5, 7)
        True
        >>> str(sl)
        '7 2 7 5 3'
        >>> sl.insert_last(9, 8)
        False
        >>> str(sl)
        '7 2 7 5 3'
        """
```

Use this page to implement the solution for the linked list question from the previous page. This page WILL be marked. If you used the previous page for the linked list solution, that's fine.

5. **[3 marks] ADTs and Efficiency** Estimated Completion: 8 minutes

Consider the following implementations of a stack:

(a) `Stack1` is implemented using a `LinkedList` like the one you worked with in the preps, lecture, and lab. As a reminder, the sole attribute of a `LinkedList` is an object that keeps track of the first `_Node` in the list. Items in `Stack1` get pushed at the **end** of its internal `LinkedList` attribute.

(b) `Stack2` uses an alternative linked list implementation called `FancyLinkedList`. A `FancyLinkedList` object has three attributes: `_first`, `_last`, and `_size`, which keep track of the first `_Node`, last `_Node`, and the list length, respectively. You can assume that `FancyLinkedList` is correctly and efficiently implemented.

Items in `Stack2` get pushed at the **end** of its internal `FancyLinkedList` attribute.

(c) `Stack3` uses a built-in Python list. Items get pushed at the **front** of Stack3's internal list.

Determine how the time for each of the stack operations grows with the input size and fill in the following table using the `Big-Oh` notation that we discussed in this course, and using `n` to denote the size of the input, just like in the preps and lectures.

Operation	Stack1	Stack2	Stack3
Push			
Pop			

6. **[6 marks] Recursion** Estimated Completion: 15 minutes

Implement the following function, which checks if a string representing either a sentence or a single word, is a palindrome. A palindrome is a string which, ignoring spaces, is identical when spelled backwards.

Here are some examples:

(a) "ewe"

(b) "anna"

(c) "borrow or rob"

(d) "taco cat"

(e) "was it a car or a cat i saw"

These are all palindromes.

RESTRICTIONS (violating any of these points will change the question and ultimately result in a grade of 0):

i. This function must be implemented using recursion.

ii. This function must not mutate the original word/sentence.

iii. You may use slicing if you find it useful, but you are NOT allowed to use the built-in reversal directly: [::-1].

iv. You are NOT permitted to use workarounds like the reversed() function, which defeat the purpose of what we are asking you to do in this question recursively.

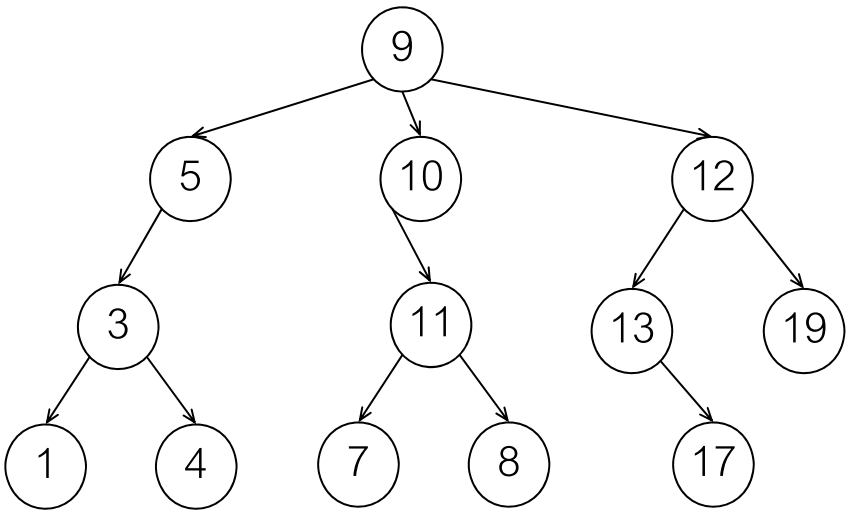
```
def ispalindrome(s: str) -> bool:
    """Return whether <s> is a palindrome.

    The function must be implemented **recursively**.
    """
```

7. **[8 marks] Trees** Estimated Completion: 15 minutes

We define the longest sequence of ascending values as the longest path in the `Tree`, starting from the root, where the values along the path are all in ascending order.

For example, in this `Tree` diagram, the longest ascending sequence is the path containing 9, 12, 13, 17, and the length of this sequence is 4.

Equal values count as well in the ascending order. For example, if the node with value 13 had value 12 instead, then the path: 9, 12, 12, 17 would still count as the longest ascending sequence.



Implement the following `Tree` method, which calculates the length of the longest ascending sequence in the `Tree`.

RESTRICTIONS (not following these restrictions will result in zero marks for this question, as you are not demonstrating knowledge on what is being tested):

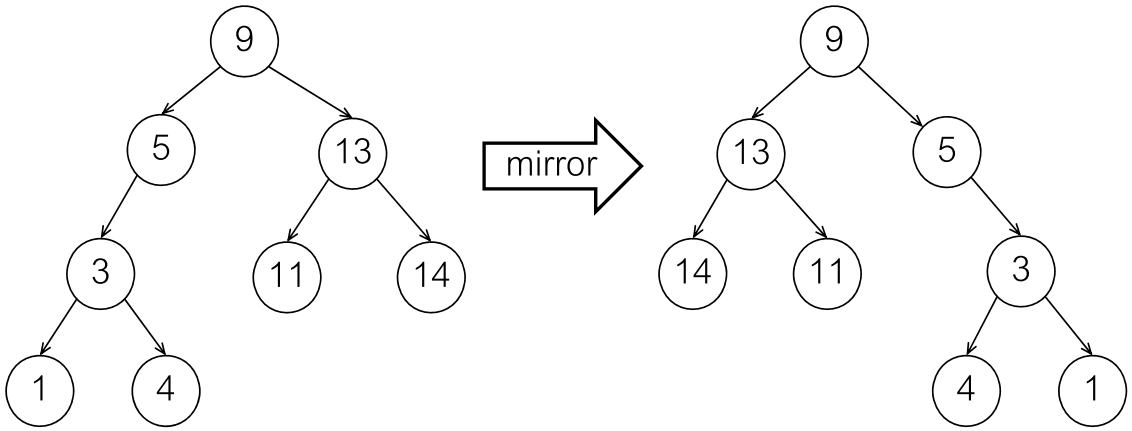
- You MUST NOT assume anything about the existence of any other methods of Tree, other than the initializer, is_empty(), and the one you have to implement below.
- You MUST NOT modify the signature of the method in any way.
- You MAY write private helper methods if you wish.
- Your method MUST USE RECURSION.
- Your method MUST NOT CAUSE INFINITE RECURSION.

```
def longest_ascending_sequence(self) -> int:
    """ Return the length of the longest ascending sequence of values
    in this Tree.
    """
```

8. **[6 marks] Binary Search Trees** Estimated Completion: 15 minutes

Implement a `BinarySearchTree` method called `mirror`, which mutates the `BinarySearchTree` into a mirror image of itself.

The diagram below is an example of the `Tree` before and after the `mirror` method is called.



Note: You do not need to worry about the fact that this method reverses the `BinarySearchTree` ordering property of the `Tree`.

RESTRICTIONS (not following these restrictions will result in zero marks for this question, as you are not demonstrating knowledge on what is being tested):

(a) This method **must be implemented using recursion.**

(b) You may declare variables but **your code must not create any new objects (no new lists, no new trees, for instance).**

(c) **No sort function allowed.**

```
def mirror(self) -> None:
    """Mutate this BinarySearchTree to transform it into a mirror image of itself.

    Hint: Consider what happens in a node and use recursion to your advantage.
    """
```

9. **[6 marks] Huffman Trees** Estimated Completion: 10 minutes

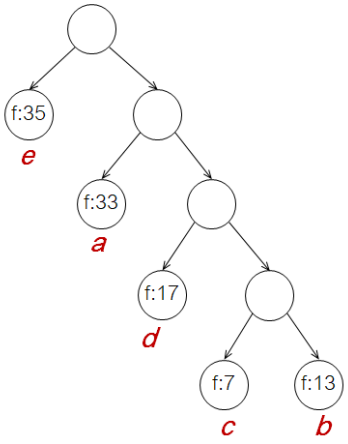
You are given the following Huffman tree and frequency table of symbols. This tree is not optimal.

a) Build the optimal tree, according to the same algorithm you used in Assignment 2.

b) Explain WHY the one we gave you below is not optimal! Show us your calculations of the size of the compressed file in each case (the given tree and the optimal one you built), to demonstrate that the given one is not optimal!

Note: Do not just regurgitate the definition of a Huffman tree, use your knowledge to answer what the questions are asking. You MUST explain WHY the one below is not optimal AND show us your calculations (NOT the calculations you did to build the tree, but the calculations of the compressed file size).

symbol	frequency
e	35
a	33
d	17
b	13
c	7



10. **[6 marks] Complexity** Estimated Completion: 15 minutes

(a) **[2 marks]** What is the time complexity of the following function, in terms of growth with input size `n`? Indicate your answer using the Θ notation, **and explain your rationale** clearly.

Note: Failure to justify your answer (or providing an incorrect explanation) will result in a grade of 0.

```
def func1(n: int) -> None:
    i, j, sum = 0, 0, 0
    while i ** 2 < n:
        while j ** 2 < n:
            sum += i * j
            j += 1
        i += 1
```

(b) **[4 marks]** What is the complexity of this function, in terms of growth with input size `n`?

Step 1: Indicate the complexity for each part of the if statement, using the Θ notation, **and explain your rationale** clearly for each part.

Step 2: Indicate the overall complexity of the function using the Θ notation, **and explain your rationale** clearly.

Note: Failure to justify your answer (or providing an incorrect explanation) will result in a grade of 0.

```
def func2(n: int) -> None:
    sum = 0
    if n % 2 == 0:
        for i in range(n // 2):
            sum += i * n
        for i in range(n // 2):
            for j in range(n ** 2):
                sum += i * j
    elif n % 3 == 0:
        sum = n**4
    else:
        while n > 1:
            n = n // 2
            sum += 1
```

Use this page for rough work. If you want work on this page to be marked, please indicate this clearly *at the location of the original question.*

Basic operators

```
True and False, True or False, not True
1 + 3, 1 - 3, 1 * 3
5 / 2 == 2.5, 5 // 2 == 2, 5 % 2 == 1
'hi' + 'bye'           # 'hibye'
[1, 2, 3] + [4, 5, 6]  # [1, 2, 3, 4, 5, 6]
```

List methods

```
lst = [1, 2, 3]
len(lst)            # 3
lst[0]              # 1
lst[0:2]            # [1, 2]
lst[0] = 'howdy'    # lst == ['howdy', 2, 3]
lst.append(29)      # lst == ['howdy', 2, 3, 29]
lst.pop()           # lst == ['howdy', 2, 3], returns 29
lst.pop(1)          # lst == ['howdy', 3], returns 2
lst.insert(1, 100)  # lst == ['howdy', 100, 3]
lst.extend([4, 5])  # lst == ['howdy', 100, 3, 4, 5]
3 in lst            # returns True
```

Dictionary methods

```
d = {'hi': 4, 'bye': 100}
d['hi']           # 4
d[100]            # raises KeyError!
'hi' in d         # True
4 in d            # False
d['howdy'] = 15   # adds new key-value pair
d['hi'] = -100    # changes a key-value pair
```

Exceptions

```
class MyCustomError(Exception):
    # Override __str__ to customize the error message.

raise MyCustomError
```

Stacks and Queues

```
s = Stack()
s.is_empty()
s.push(10)
s.pop()  # Raises an EmptyStackError if stack is empty.

q = Queue()
q.is_empty()
q.enqueue(10)
q.dequeue()  # Returns None if queue is empty.
```

Class Design

```
class SomeClass:
    """ Class description

    Description of Public Attributes.
    """
    # Description of Private Attributes
    # Type Contract for public and private attributes

    # Public and private methods follow the
                # function design recipe from CSC108.

sc = SomeClass(...)   # create an instance
```

Linked Lists

```
class _Node:
    """A node in a linked list.

    === Attributes ===
    item: The data stored in this node.
    next:
        The next node in the list, or None if there are
        no more nodes in the list.
    """
    item: Any
    next: Optional[_Node]

    def __init__(self, item: Any) -> None:
        """Initialize a new node storing <item>,
        with no 'next' node.
        """

class LinkedList:
    """A linked list implementation of the List ADT.
    """
    # === Private Attributes ===
    # _first:
    #     The first node in the linked list,
    #     or None if the list is empty.
    _first: Optional[_Node]

    def __init__(self) -> None:
        """Initialize an empty linked list."""

    # alternative initializer, using a Python list
    def __init__(self, items: list) -> None:
        """Initialize a linked list with the given items.

        The first node in the linked list contains the
        first item in <items>.
        """
```

Summation identities

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

Trees

```
class Tree:
    # === Private Attributes ===
    # _root: The item stored at this tree's root, or None if this tree is empty.
    _root: Optional[Any]
    # _subtrees: The list of all subtrees of this tree.
    _subtrees: List[Tree]

    # === Representation Invariants ===
    # - If self._root is None then self._subtrees is an empty list.
    #   This setting of attributes represents an empty tree.
    # - self._subtrees does not contain any empty trees.

    def __init__(self, root: Optional[Any], subtrees: List[Tree]) -> None:
        """Initialize a new Tree with the given root value and subtrees.

        If <root> is None, the tree is empty.
        Precondition: if <root> is None, then <subtrees> is empty.
        """

    def is_empty(self) -> bool:
        """Return whether this tree is empty."""
```

Binary Search Trees

```
class BinarySearchTree:
    # === Private Attributes ===
    # _root: The item stored at the root of the tree, or None if the tree is empty.
    _root: Optional[Any]
    # _left: The left subtree, or None if the tree is empty.
    _left: Optional[BinarySearchTree]
    # _right: The right subtree, or None if the tree is empty.
    _right: Optional[BinarySearchTree]

    # === Representation Invariants ===
    #  - If self._root is None, then so are self._left and self._right.
    #    This represents an empty BST.
    #  - If self._root is not None, then self._left and self._right are BinarySearchTrees.
    #  - (BST Property) If self is not empty, then
    #    all items in self._left are <= self._root, and
    #    all items in self._right are >= self._root.

    def __init__(self, root: Optional[Any]) -> None:
        """Initialize a new BST containing only the given root value.

        If <root> is None, initialize an empty tree.
        """
```

Abstract Syntax Trees

```
class Statement:
    """An abstract class representing a Python statement."""
    def evaluate(self, env: Dict[str, Any]) -> Optional[Any]:
        """Evaluate this statement with the given environment."""

class Expr(Statement):
    """An abstract class representing a Python expression."""
```