

1. [6 marks] **Exceptions** Consider the following function:

```
def solve(x, y):
    try:
        if x > 50:
            raise ValueError
        a = x / y
        solution = 2 * a
        print(solution)
    except ZeroDivisionError:
        print('Cannot divide by zero')
    except NameError: # this error is raised when a variable that has not been defined is used
        print('Name not defined inside')
    except:
        print('Something is wrong')
    finally:
        print('End of program')
```

What would be the output produced by each of the segments of code below?

```
>>> solve(20, 2)
```

```
>>> solve(100, 50)
```

```
>>> solve('science', 1)
```

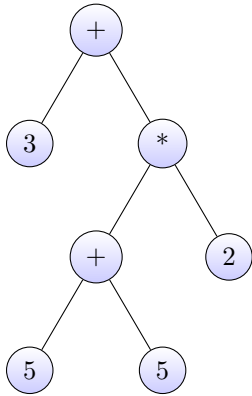
```
>>> try:
...     solve(x, 1)
... except NameError:
...     print('Name not defined outside')
```

```
>>> try:
...     print(solve(5, 1) * 2)
... except:
...     print('Something is wrong outside')
```

2. [4 marks] **Tree Traversals**

For this question, we will be dealing with binary trees in which each internal node corresponds to a simple mathematical operator (one of +, -, * or /) and each leaf node corresponds to an integer operand.

For example, the tree representation for $3 + ((5+5)*2)$ would be:



- (a) [2 marks] If I wanted to traverse through such a tree to return the expression it represents in the form of a string such as $3 + ((5+5)*2)$ for the example tree above, which of the following would be most suitable? Explain your answer. **No explanation = no marks**

- i. In-order traversal
- ii. Pre-order traversal
- iii. Post-order traversal
- iv. Level order traversal

- (b) [2 marks] If I wanted to traverse through such a tree to evaluate the expression and return the solution, such as for the example tree above, we would return the integer 23, then which of the above traversal algorithms would be most suitable? Explain your answer. **No explanation = no marks**

3. [4 marks] Complete the following `BinarySearchTree` method according to the docstring (each docstring example includes an associated image beside it of the tree with all items that need to be removed highlighted in blue).

Hint 1: Don't forget to ensure that all of your representation invariants are satisfied before the method returns.

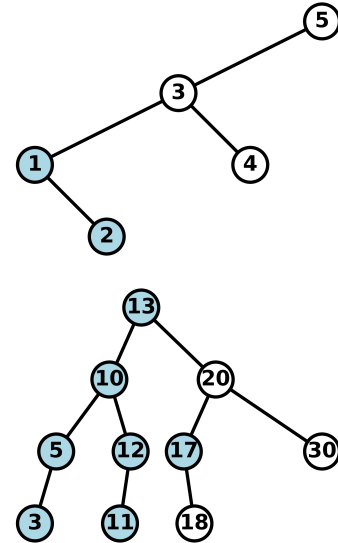
Hint 2: You should find the “promotion” strategy from when we covered BST deletion to be helpful.

Note: See Aid sheet for BST documentation.

```
def prune(self, low: int) -> None:
    """
    Remove all values from this BST that are less than <low>

    Precondition: all values in this BST are integers

    >>> root = BinarySearchTree(None)
    >>> for x in [5, 3, 4, 1, 2]:
    ...     root.insert(x)
    >>> root.prune(3)
    >>> assert root.items() == [3, 4, 5]
    >>> me = BinarySearchTree(None)
    >>> for x in [13, 20, 17, 30, 18, 10, 5, 12, 11, 3]:
    ...     me.insert(x)
    >>> me.prune(18)
    >>> assert me.items() == [18, 20, 30]
    """
```



4. [6 marks] Stacks / Recursion

- (a) [4 marks] Complete the following function which takes an item and inserts it at the bottom of a stack.

Note: See Aid sheet for Stack API.

RESTRICTIONS (violating any of these points will result in a grade of 0):

- i. Your function must be **recursive**.
- ii. You may not use any loops.
- iii. The Stack class only has the methods `is.empty`, `push` and `pop`.
- iv. You may not create any new objects (no new stacks, lists, etc., including temporary ones).

```
def insert_at_bottom(s: Stack, item: Any) -> None:
    """Insert the given <item> at the bottom of the given Stack <s>."""

    if s.is_empty():

    else:
```

- (b) [2 marks] Complete the type contract by filling in the return type, and then write a proper docstring description of the function below (which uses `insert_at_bottom` from the previous question as a helper).

Your description must be clear and concise, and follow proper docstring style (do **not** give a line-by-line description of the code).

```
def mystery(s: Stack) -> ____: # FILL IN RETURN TYPE HERE
    """
    # YOUR DOCSTRING DESCRIPTION HERE

    """

    if not s.is_empty():
        temp = s.pop()
        mystery(s)
        insert_at_bottom(s, temp)
```