Question **1**

Correct

Mark 1.00 out of 1.00

Given the AST declarations as follows:

class Program: #decl:List[VarDecl],stmts:List[Stmt]

class VarDecl: #name:str

class Stmt(ABC): #abstract class

class Block(Stmt): #decl:List[VarDecl],stmts:List[Stmt]

class Assign(Stmt): #lhs:Id,rhs:Exp

class Exp(ABC): #abstract class

class BinOp(Exp): #op:str,e1:Exp,e2:Exp #op is +,-,*,/,+.,-.,*.,/., &&,||, >, >., >b, =, =., =b

class UnOp(Exp): #op:str,e:Exp #op is -,-., !,i2f, floor

class IntLit(Exp): #val:int

class FloatLit(Exp): #val:float

class BoolLit(Exp): #val:bool

class Id(Exp): #name:str

and the Visitor class is declared as follows:

class StaticCheck(Visitor):

    def visitProgram(self,ctx:Program,o):pass

    def visitVarDecl(self,ctx:VarDecl,o): pass

    def visitBlock(self,ctx:Block,o): pass

    def visitAssign(self,ctx:Assign,o): pass

    def visitBinOp(self,ctx:BinOp,o): pass

    def visitUnOp(self,ctx:UnOp,o):pass

    def visitIntLit(self,ctx:IntLit,o): pass

    def visitFloatLit(self,ctx,o): pass

    def visitBoolLit(self,ctx,o): pass

    def visitId(self,ctx,o): pass

Rewrite the body of the methods in class StaticCheck to infer the type of identifiers and check the following type constraints:

- + , - , *, / accept their operands in int type and return int type
- +., -., *., /. accept their operands in float type and return float type
- > and = accept their operands in int type and return bool type
- >. and =. accept their operands in float type and return bool type
- !, &&, ||, >b and =b accept their operands in bool type and return bool type
- i2f accepts its operand in int type and return float type
- floor accept its operand in float type and return int type
- In an assignment statement, the type of lhs must be the same as that of rhs, otherwise, the exception TypeMismatchInStatement should be raised together with the assignment statement.
- the type of an Id is inferred from the above constraints in the first usage,
    - if the Id is not in the declarations, exception UndeclaredIdentifier should be raised together with the name of the Id, or
    - If the Id cannot be inferred in the first usage, exception TypeCannotBeInferred should be raised together with the assignment statement which contains the type-unresolved identifier.
- For static referencing environment, this language applies the scope rules of block-structured programming language. When there is a declaration duplication of a name in a scope, exception Redeclared should be raised together with the second declaration.

- If an expression does not conform the type constraints, the StaticCheck will raise exception TypeMismatchInExpression with the expression.

Your code starts at line 110

For example:

Test	Result
`Program([VarDecl("x")],[Assign(Id("x"),IntLit(3)),Block([VarDecl("y")],[Assign(Id("x"),Id("y")),Assign(Id("y"),BoolLit(True))])])`	`Type Mismatch In Statement: Assign(Id("y"),BoolLit(True))`

Answer: (penalty regime: 0 %)

Ace editor not ready. Perhaps reload page?
Falling back to raw text area.

```
INT = 1
BOOL = 2
FLOAT = 3


class Variable:
    def __init__(self):
        self._type = None

    @property
    def rt(self):
        return self._type

    @rt.setter
    def rt(self, RT):
        if self._type is None:
            self._type = RT
```

	Test	Expected
✓	`Program([VarDecl("x")],[Assign(Id("x"),IntLit(3)),Block([VarDecl("y")],[Assign(Id("x"),Id("y")),Assign(Id("y"),BoolLit(True))])])`	`Type Mismatch` `Assign(Id("y")`
✓	`Program([VarDecl("x")],[Assign(Id("x"),IntLit(3)),Block([VarDecl("y"),VarDecl("x"),VarDecl("y")],[Assign(Id("x"),Id("y")),Assign(Id("y"),IntLit(3))])])`	`Redeclared: Va`
✓	`Program([VarDecl("x")],[Assign(Id("x"),IntLit(3)),Block([VarDecl("y"),VarDecl("x")],[Assign(Id("x"),Id("y")),Assign(Id("y"),FloatLit(3))])])`	`Type Cannot Be`
✓	`Program([VarDecl("x"),VarDecl("t")],[Assign(Id("x"),IntLit(3)),Block([VarDecl("y")],[Assign(Id("x"),Id("y")),Block([],[Assign(Id("t"),FloatLit(3)),Assign(Id("z"),Id("t"))])])])`	`Undeclared Ide`
✓	`Program([VarDecl("x"),VarDecl("t")],[Assign(Id("x"),IntLit(3)),Block([VarDecl("y")],[Assign(Id("x"),Id("y")),Block([VarDecl("z")],[Assign(Id("t"),FloatLit(3)),Assign(Id("z"),UnOp("-",Id("t")))])])])`	`Type Mismatch`
✓	`Program([VarDecl("x"),VarDecl("t")],[Assign(Id("x"),IntLit(3)),Block([VarDecl("y")],[Assign(Id("x"),Id("y")),Block([VarDecl("z")],[Assign(Id("t"),FloatLit(3)),Assign(Id("z"),BinOp("-",Id("t"),Id("x")))])])])`	`Type Mismatch`

	Test	Expected
✓	`Program([VarDecl("x"),VarDecl("t")],[Assign(Id("x"),IntLit(3)),Block([VarDecl("y")],[Assign(Id("x"),Id("y")),Block([VarDecl("z")],[Assign(Id("t"),FloatLit(3)),Assign(Id("y"),BinOp("-.",Id("t"),UnOp("i2f",Id("x"))))])])])`	`Type Mismatch I` `Assign(Id("y")`
✓	`Program([VarDecl("x"),VarDecl("t")],[Assign(Id("x"),IntLit(3)),Block([VarDecl("y")],[Assign(Id("x"),Id("y")),Block([VarDecl("z")],[Assign(Id("t"),FloatLit(3)),Assign(Id("z"),UnOp("floor",Id("y")))])])])`	`Type Mismatch I`
✓	`Program([VarDecl("x"),VarDecl("t")],[Assign(Id("x"),IntLit(3)),Block([VarDecl("x")],[Assign(Id("x"),FloatLit(3.0)),Assign(Id("t"),Id("x"))]),Assign(Id("x"),Id("t"))])`	`Type Mismatch I`
✓	`Program([VarDecl("x")],[Assign(Id("x"),IntLit(3)),Block([VarDecl("x")],[Assign(Id("x"),FloatLit(3.0))]),Assign(Id("x"),BoolLit(False))])`	`Type Mismatch I` `Assign(Id("x")`

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

Question **2**

Correct

Mark 1.00 out of 1.00

Given the AST declarations as follows:

class Program: #decl:List[Decl],stmts:List[Stmt]

class Decl(ABC): #abstract class

class VarDecl(Decl): #name:str

class FuncDecl(Decl): #name:str,param:List[VarDecl],local:List[Decl],stmts:List[Stmt]

class Stmt(ABC): #abstract class

class Assign(Stmt): #lhs:Id,rhs:Exp

class CallStmt(Stmt): #name:str,args:List[Exp]

class Exp(ABC): #abstract class

class IntLit(Exp): #val:int

class FloatLit(Exp): #val:float

class BoolLit(Exp): #val:bool

class Id(Exp): #name:str

and the Visitor class is declared as follows:

```
class StaticCheck(Visitor):
    def visitProgram(self,ctx:Program,o):pass
    def visitVarDecl(self,ctx:VarDecl,o): pass
    def visitFuncDecl(self,ctx:FuncDecl,o): pass
    def visitCallStmt(self,ctx:CallStmt,o):pass
    def visitAssign(self,ctx:Assign,o): pass
    def visitIntLit(self,ctx:IntLit,o): pass
    def visitFloatLit(self,ctx,o): pass
    def visitBoolLit(self,ctx,o): pass
    def visitId(self,ctx,o): pass
```

Rewrite the body of the methods in class StaticCheck to infer the type of identifiers and check the following type constraints:

- In an Assign, the type of lhs must be the same as that of rhs, otherwise, the exception TypeMismatchInStatement should be raised together with the Assign
- the type of an Id is inferred from the above constraints in the first usage,
    - if the Id is not in the declarations, exception UndeclaredIdentifier should be raised together with the name of the Id, or
    - If the Id cannot be inferred in the first usage, exception TypeCannotBeInferred should be raised together with the statement
- For static referencing environment, this language applies the scope rules of block-structured programming language where a function is a block. When there is a declaration duplication of a name in a scope, exception Redeclared should be raised together with the second declaration.
- In a call statement, the argument type must be the same as the parameter type. If there is no function declaration in the static referencing environment, exception UndeclaredIdentifier should be raised together with the function call name. If the numbers of parameters and arguments are not the same or at least one argument type is not the same a the type of the corresponding parameter, exception TypeMismatchInStatement should be raise with the call statement If there is at least one parameter type cannot be resolved, exception TypeCannotBeInferred should be raised together with the call statement.

Your code starts at line 120

For example:

Test	Result
`Program([VarDecl("x"),FuncDecl("foo",[VarDecl("x")],[],[Assign(Id("x"),FloatLit(2))])],[Assign(Id("x"),IntLit(3)),CallStmt("foo",[Id("x")])])`	`Type Mismatch In Statement: CallStmt("foo",[Id("x")])`

Answer: (penalty regime: 0 %)

Ace editor not ready. Perhaps reload page?
Falling back to raw text area.

```
INT = 1
BOOL = 2
FLOAT = 3


class Variable:
    def __init__(self):
        self._type = None

    @property
    def rt(self):
        return self._type

    @rt.setter
    def rt(self, RT):
        if self._type is None:
            self._type = RT
```

	Test	Expected	Got	
✓	`Program([VarDecl("x"),FuncDecl("foo",[VarDecl("x")],[],[Assign(Id("x"),FloatLit(2))])],[Assign(Id("x"),IntLit(3)),CallStmt("foo",[Id("x")])])`	`Type Mismatch In Statement: CallStmt("foo",[Id("x")])`	`Type Mismatch In Statement: CallStmt("foo",[Id("x")])`	✓
✓	`Program([VarDecl("x"),FuncDecl("foo",[],[VarDecl("x")],[Assign(Id("x"),FloatLit(2))])],[Assign(Id("x"),IntLit(3)),CallStmt("foo",[Id("x")])])`	`Type Mismatch In Statement: CallStmt("foo",[Id("x")])`	`Type Mismatch In Statement: CallStmt("foo",[Id("x")])`	✓
✓	`Program([VarDecl("x"),FuncDecl("x",[VarDecl("y")],[],[])],[])`	`Redeclared: FuncDecl(x, [VarDecl("y")],[],[])`	`Redeclared: FuncDecl(x, [VarDecl("y")],[],[])`	✓
✓	`Program([VarDecl("x"),FuncDecl("foo",[VarDecl("y")],[],[CallStmt("x",[])])],[])`	`Undeclared Identifier: x`	`Undeclared Identifier: x`	✓
✓	`Program([VarDecl("x"),FuncDecl("foo",[VarDecl("y")],[FuncDecl("foo2",[],[],[])],[CallStmt("foo2",[])])],[CallStmt("foo2",[])])`	`Undeclared Identifier: foo2`	`Undeclared Identifier: foo2`	✓
✓	`Program([VarDecl("x"),FuncDecl("foo",[VarDecl("y")],[],[])],[CallStmt("foo",[IntLit(3)]),CallStmt("foo",[Id("x")]),Assign(Id("x"),FloatLit(0.0))])`	`Type Mismatch In Statement: Assign(Id("x"),FloatLit(0.0))`	`Type Mismatch In Statement: Assign(Id("x"),FloatLit(0.0))`	✓

	Test	Expected	Got	
✓	`Program([VarDecl("x"),FuncDecl("foo",[VarDecl("y"),VarDecl("z")],[],[Assign(Id("z"),FloatLit(0.0))])],[CallStmt("foo",[IntLit(3),Id("x")]),CallStmt("foo",[Id("x"),FloatLit(0.0)])])`	`Type Mismatch In Statement: CallStmt("foo",[Id("x"),FloatLit(0.0)])`	`Type Mismatch In Statement: CallStmt("foo",[Id("x"),FloatLit(0.0)])`	✓
✓	`Program([VarDecl("x"),FuncDecl("foo",[VarDecl("y"),VarDecl("z")],[],[])],[CallStmt("foo",[IntLit(3),Id("x")])])`	`Type Cannot Be Inferred: CallStmt("foo",[IntLit(3),Id("x")])`	`Type Cannot Be Inferred: CallStmt("foo",[IntLit(3),Id("x")])`	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.