Analsis:

Diferencias entre Visitor y Listener

Visitor se implementan nuestras reglas de visita, en que orden se realizan las visitas y como. Es un recorrido manual y controlado, si no se llama .visit() dentro de un hijo, su subárbol no se procesa. El flujo y profundidad de visita del árbol depende de la implementación. Usualmente suele parar cuando encuentra el primer error.

Listerner recorre el arbol de forma automática, el llamando los métodos de forma enterX y enterY y solo se tiene que hacer un lógica después de procesar una suma / resta.

Para el archivo program_test_pass.txt

Visitor y Lister pasan sin ningún problema porque no presentan ninguna error

```
appuser@7de809003098:/program$ python3 Driver.py program_test_pass.txt
Type checking passed
appuser@7de809003098:/program$ python3 DriverListener.py program_test_pass.txt
Type checking passed
```

Para el archivo program_test_no_pass.txt

Visitor pasa porque al no asegurarnos de visitar todos los nodos y algunos errores nunca se procesan.

Lister detecta los fallos al recorrer el árbol completo. además este guarda los errores y tipos para poder imprimir de mejor forma los errores.

```
appuser@7de809003098:/program$ python3 Driver.py program_test_no_pass.txt
Type checking passed
appuser@7de809003098:/program$ python3 DriverListener.py program_test_no_pass.txt
Type checking error: Unsupported operand types for * or /: int and string
Type checking error: Unsupported operand types for * or /: int and string
Type checking error: Unsupported operand types for + or -: float and bool
Type checking error: Unsupported operand types for + or -: string and int
```

Extender la gramática

Se añadió modulo y se añadió elevación

```
expr: expr '**' expr          # Pow
    | expr op=('*'|'/'|'%') expr       # MulDiv
```

Listerner

```
def exitPow(self, ctx: SimpleLangParser.PowContext):
    left_type = self.types[ctx.expr(0)]
    right_type = self.types[ctx.expr(1)]
    if not self.is_valid_arithmetic_operation(left_type, right_type):
        self.errors.append(f"Unsupported operand types for **: {left_type} and {right_type}")
    self.types[ctx] = FloatType() if isinstance(left_type, FloatType) or isinstance(right_type, FloatType) else IntTyp
```

Visitor

```
def visitPow(self, ctx: SimpleLangParser.PowContext):
    left_type = self.visit(ctx.expr(0))
    right_type = self.visit(ctx.expr(1))        You, 18 minutes ago • Uncommitted changes
    if isinstance(left_type, (IntType, FloatType)) and isinstance(right_type, (IntType, FloatType)):
        return FloatType() if isinstance(left_type, FloatType) or isinstance(right_type, FloatType) else IntType()
    else:
        raise TypeError(f"Unsupported operand types for **: {left_type} and {right_type}")
```

Extender el sistema de tipo :

- Se añadió la división por 0
- Se añadió la suma entre string
- Cualquier operación que se intentara hacer con un boolean se retorna error
- Se valido que para modulo solo fuera INT % INT
- Se valido que para la elevación únicamente se use INT o Float