Fun Language Exercise

In this exercise, you will implement an interpreter for the *Fun* programming language. Fun is *embedded* in Python, meaning Fun programs are a subset of Python objects.

- **Fun Int**: Every `int` object is a Fun program. (examples: `1`, `-2`)
- The Python string `"+"` is a Fun keyword.
- **Fun Plus**: If `x` and `y` are Fun programs, then the tuple `("+", x, y)` is also a Fun program. (examples: `("+", 1, 2)`, `("+", 1, ("+", 2, 3))`)

Task 1: Implement a function `evaluate(program)` that *evaluates* a Fun program. The evaluation of an integer is itself, and the evaluation of a tuple `("+", x, y)` is the evaluation of `x` plus the evaluation of `y`. (examples: `evaluate(1)` returns `1`, `evaluate(("+", 1, ("+", 2, 3)))` returns `6`).

We extend the Fun language as follows:

- `"fun"` is also a Fun keyword.
- **Fun function**: If `p` is a Python string that is not a Fun keyword and `retval` is a Fun program, then the tuple `("fun", p, retval)` is also a Fun program. This Fun program represents an anonymous function (lambda) that receives a single parameter `p` and returns the value `retval`. (example: `("fun", "x", ("+", "x", 1))`)
- **Fun Var**: A Python string that is not a Fun keyword is a Fun program called a Fun variable. (examples: `"x"`, `"ABC"`)
- **Fun Call**: If `foo` is a Fun anonymous function and `arg` is a Fun program, then the tuple `(foo, arg)` is also a Fun program. (example: `(("fun", "x", ("+", "x", 1)), 2)`)

Task 2: Extend the `evaluate` function from Task 1 to support the new Fun programs. If `foo=("fun", p, retval)` is a Fun function and `arg` is a Fun program, then the evaluation of the Fun call `(foo, arg)` is the evaluation of `retval` where each instance of `p` is replaced by `arg`.

Note that Fun functions only receive and return integer values.

Some examples:

```
identity = ("fun", "x", "x")
identity_of_3 = (identity, 3)
print(evaluate(identity_of_3))  # 3

add_3 = ("fun", "x", ("+", 3, "x"))
print(evaluate((add_3, 1)))  # 4
print(evaluate((add_3, 5)))  # 8
```

Task 3: Implement a Visitor interface for Fun programs.

Note: Don't feel forced to stick to the standard Visitor design pattern. Do whatever it takes to support a Visitor interface that makes sense.

For example, the following visitor logs all integer values in the program:

```
class IntLogger(FunVisitor):
  def visit_int(self, val):
    print(val)

program = ("+", 1, ("+", 2, 3))
IntLogger().visit(program)  # prints 1, 2, 3
```

Task 4: Implement the following visitors:

- A visitor that raises an error if the program contains an unbound variable (i.e., a variable `p` that's *not* inside a Fun function `("fun", p, ...)`).
- A visitor that raises an error if the program contains a function that returns a function instead of an integer.
- A visitor that raises an error if the program contains a function call where the argument is a function instead of an integer.
- Re-implement the `evaluate` function as a Fun visitor. (Hint: Support `visit` methods that return a value.)

Finally, implement a function `evaluate` that runs the visitors above in a sequence and returns the result of the evaluator visitor (last bullet).

Task 5: Add tests.