This exam has 3 questions, for a total of 10 points.

1. **3 points** Draw the abstract syntax tree that Python’s builtin parser would produce for the following program.

   ```
   x = input()
y = x
print - 3+y
   ```

   **Solution:**

   ![Abstract Syntax Tree](image)

   Solution:

2. **3 points** In \( P_0 \), the following language constructs are *expressions*: integer constants, variables, addition, unary subtraction, and calls to the input function. A *simple expression* is a variable or constant. A *complex expression* is an expression with one or more children that are not simple. Flatten the program given in question 1 by changing it to an equivalent \( P_0 \) program (produces the same output) that does not contain any complex expressions. Also, make sure that the argument to `print` is a simple expression.

   **Solution:**
```python
x = input()
y = x
tmp0 = -3
tmp1 = tmp0 + y
print tmp1
```
3. **4 points** Fill in the missing code for `UnarySub` in the definition of the following `flatten` function.

```python
def flatten(n, needs_to_be_simple):
    if isinstance(n, Module):
        return Module(n.doc, flatten(n.node, False))
    elif isinstance(n, Stmt):
        sss = [flatten(s, False) for s in n.nodes]
        return Stmt(reduce(lambda a,b: a + b, sss, []))
    elif isinstance(n, Assign):
        (rhs, ss) = flatten(n.expr, False)
        return ss + [Assign(n.nodes, rhs)]
    elif isinstance(n, Name):
        return (n, [])
    elif isinstance(n, Add):
        (left, ss1) = self.dispatch(n.left, True)
        (right, ss2) = self.dispatch(n.right, True)
        if needs_to_be_simple:
            tmp = generate_name('tmp')
            assign = Assign(nodes=[AssName(name=tmp, flags='OP_ASSIGN')],
                             expr=Add((left, right)))
            return (Name(tmp), ss1 + ss2 + [assign])
        else:
            return (Add((left, right)), ss1 + ss2)
    elif isinstance(n, UnarySub):
        (expr, ss) = flatten(n.expr, True)
        if needs_to_be_simple:
            tmp = generate_name('tmp')
            assign = Assign(nodes=[AssName(name=tmp, flags='OP_ASSIGN')],
                             expr=UnarySub(expr))
            return (Name(tmp), ss + [assign])
        else:
            return (UnarySub(expr), ss)

elif ...
```

**Solution:**

```python
(expr, ss) = flatten(n.expr, True)
if needs_to_be_simple:
    tmp = generate_name('tmp')
    assign = Assign(nodes=[AssName(name=tmp, flags='OP_ASSIGN')],
                     expr=UnarySub(expr))
    return (Name(tmp), ss + [assign])
else:
    return (UnarySub(expr), ss)
```