Execution rule for while statements:

Evaluation rule for not expressions:

Evaluation rule for and expressions:

Evaluation rule for or expressions:

Execution rule for assignment statements:

Execution rule for def statements:

Applying user-defined functions:

Evaluation rule for call expressions:

Expressions (Program):

Environments (Names):

Frames link to each other. An environment is a sequence of frames. An environment is a first frame, plus the environment that follows.

Evaluation rule for call expressions:

Applies user-defined functions:

1. Create a new local frame that extends the environment with which the function is associated.
2. Bind the arguments to the function's formal parameter names in that frame.
3. Execute the body of the function in the environment beginning at that frame.

Execution rule for def statements:

1. Create a new function value with the specified name, formal parameters, and function body.
2. Associate that function with the current environment.
3. Bind the name of the function to the function value in the first frame of the current environment.

Execution rule for assignment statements:

1. Evaluate the expression(s) on the right of the equal sign.
2. Simultaneously bind the names on the left to those values in the first frame of the current environment.

Execution rule for conditional statements:

Each clause is considered in order.

1. Evaluate the header's expression.
2. If it is a true value, execute the suite, then skip the remaining clauses in the statement.

Evaluation rule for for expressions:

1. Evaluate the subexpression <left>.
2. If the result is a true value v, then the expression evaluates to v.
3. Otherwise, the expression evaluates to the value of the subexpression <right>.

Evaluation rule for and expressions:

1. Evaluate the subexpression <left>.
2. If the result is a false value v, then the expression evaluates to v.
3. Otherwise, the expression evaluates to the value of the subexpression <right>.

Evaluation rule for or expressions:

1. Evaluate the subexpression <left>.
2. If the result is a true value v, then the expression evaluates to v.
3. Otherwise, the expression evaluates to the value of the subexpression <right>.

Evaluation rule for not expressions:

1. Evaluate <exp>; The value is True if the result is a false value, and False otherwise.

Execution rule for while statements:

1. Evaluate the header's expression.<left>.
2. If it is a true value, execute the (whole) suite, then return to step 1.
What does `sum_squares` need to know about `square`?

Square computes the square of a number.

Data abstraction: A methodology by which functions enforce an abstraction barrier between representation and use.

```python
square = lambda x, y: x * y

A function
with formal parameters x and y
and body “return x * y”

Must be a single expression
```

```python
def make_adder(n):
    """Return a function that takes one argument k and returns k + n.
    """
    def adder(k):
        return k + n
    return adder

make_adder: n: 1
adder: ...
make_adder: k: 2

A local def statement
Can refer to names in the enclosing function
```

```python
def square(x):
    def square(x):
        return x ** 2

What does `sum_squares` need to know about `square`?

Square computes the square of a number. Yes Square computes the square by calling `mul`.

Rational numbers in the problem domain

```
add_rat mul_rat eq_rat
```

Rational numbers as numerators & denominators

```
make_rat numer denom
```

Rational numbers as tuples

```
tuplegetitem
```

Three numeric types in Python:

```python
>>> type(2)
<

Represents integers exactly
```

```python
>>> type(1.5)
<class 'float'>

Represents real numbers approximately
```

```python
>>> type(1+1j)
<class 'complex'>

A function is associated with the first frame of the environment in which the function was defined
```

```python
@trace
def triple(x):
    return 3 * x

def triple(x):
    return 3 * x
triple = trace(triple)

@trace
def square(x):
    return x ** 2

def square(x):
    return x ** 2

square = lambda x: x ** 2

def square(x):
    def square(x):
        return x ** 2

What does `sum_squares` need to know about `square`?

Square computes the square of a number. Yes Square computes the square by calling `mul`. No

```
def make_adder(n):
    """Return a function that takes one argument k and returns k + n.
    """
    def adder(k):
        return k + n
    return adder

make_adder: n: 1
adder: ...
make_adder: k: 2

A local def statement
Can refer to names in the enclosing function
```

```python
square = lambda x: x ** 2
```

```python
def square(x):
    def square(x):
        return x ** 2

What does `sum_squares` need to know about `square`?

Square computes the square of a number. Yes Square computes the square by calling `mul`. No

```python
>>> square = lambda x, y: x * y
>>> square(3, 4)
12
```

```python
Rational numbers as numerators & denominators
```

```
make_rat numer denom
```

Rational numbers as tuples

```
tuplegetitem
```

Three numeric types in Python:

```python
>>> type(2)
<

Represents integers exactly
```

```python
>>> type(1.5)
<class 'float'>

Represents real numbers approximately
```

```python
>>> type(1+1j)
<class 'complex'>

A function is associated with the first frame of the environment in which the function was defined
```

```python
def square(x):
    def square(x):
        return x ** 2

What does `sum_squares` need to know about `square`?

Square computes the square of a number. Yes Square computes the square by calling `mul`.

```
def make_adder(n):
    """Return a function that takes one argument k and returns k + n.
    """
    def adder(k):
        return k + n
    return adder

make_adder: n: 1
adder: ...
make_adder: k: 2

A local def statement
Can refer to names in the enclosing function
```

```python
square = lambda x: x ** 2
```

```python
def square(x):
    def square(x):
        return x ** 2

What does `sum_squares` need to know about `square`?

Square computes the square of a number. Yes Square computes the square by calling `mul`. No

```python
>>> square = lambda x, y: x * y
>>> square(3, 4)
12
```

```python
Rational numbers as numerators & denominators
```

```
make_rat numer denom
```

Rational numbers as tuples

```
tuplegetitem
```

Three numeric types in Python:

```python
>>> type(2)
<

Represents integers exactly
```

```python
>>> type(1.5)
<class 'float'>

Represents real numbers approximately
```

```python
>>> type(1+1j)
<class 'complex'>

A function is associated with the first frame of the environment in which the function was defined
```

```python
def square(x):
    def square(x):
        return x ** 2

What does `sum_squares` need to know about `square`?

Square computes the square of a number. Yes Square computes the square by calling `mul`.

```
def make_adder(n):
    """Return a function that takes one argument k and returns k + n.
    """
    def adder(k):
        return k + n
    return adder

make_adder: n: 1
adder: ...
make_adder: k: 2

A local def statement
Can refer to names in the enclosing function
```

```python
square = lambda x: x ** 2
```