The Elements of Programming

- Primitive Expressions and Statements
  - The simplest building blocks of a language

- Means of Combination
  - Compound elements are built from simpler ones

- Means of Abstraction
  - Compound elements can be named and manipulated as units

Programming languages allow us to communicate, too

Functions and Data

**Data:** Stuff we want to manipulate

“The Art of Computer Programming”

Donald Knuth
(Ka-Noot)

This slide

**Functions:** Rules for manipulating data

1. Add numbers
2. Count the words in a line of text
3. Pronounce someone’s name
4. Load the next slide
5. Pronounce someone’s name

Types of expressions

An expression describes a computation and evaluates to a value

- \( 18 + 69 \)
- \( \frac{6}{23} \)
- \( \sin \pi \)
- \( \sqrt{3493161} \)
- \( \sum_{i=1}^{100} i \)
- \( | - 1869| \)
- \( \left( \frac{69}{18} \right) \)

Call Expressions in Python

All expressions can use function call notation

(Demo)

Anatomy of a Call Expression

\[
\begin{aligned}
\text{add} & \quad ( \quad 2 \quad , \quad 3 \quad ) \\
\text{Operator} & \quad \text{Operand 0} \quad \text{Operand 1}
\end{aligned}
\]

Operators and operands are expressions

So they evaluate to values

Evaluation procedure for call expressions:

1. Evaluate the operator and operand subexpressions
2. Apply the function that is the value of the operator subexpression to the arguments that are the values of the operand subexpression
Evaluating Nested Expressions

\[ \text{mul}(\text{add}(2, \text{mul}(4, 6)), \text{add}(3, 5)) \]

208

\[ \text{add}(2, \text{mul}(4, 6)) \]

26

\[ \text{mul}(4, 6) \]

8

\[ \text{add}(3, 5) \]

The Print Function

\((\text{Demo})\)

Nested Expressions with Print

Pure Functions & Non-Pure Functions

Pure Functions

Function signature: how many parameters

-2

\(\text{abs}(\text{number})\):

2

Only produces return values

2, 100

\(\text{pow}(x, y)\):

1267650608228229401496783205376

Non-Pure Functions

-2

\(\text{print}(\text{...})\):

None

May create side effects

\(\text{display} \text{"-2"}\)

Names and Assignment

\(\text{abs}(\text{x}):\)

\(\text{max}(\text{a}, \text{b}, \text{c}, \text{...}):\)

\(\text{pi} :\)

3.14...

Built-in function

Imported value

Assigned value

\(\text{from math import pi}\)

\(\text{tau = 2 * pi}\)

The environment does not track where names came from
Environments

A frame holds name bindings

max:
... 
max(a, b, c, ...):

pi: 3.14 
tau: 6.28

from math import pi
tau = 2 * pi

The environment does not track where names came from

User-Defined Functions

Named values are a simple means of abstraction

Named expressions are a more powerful means of abstraction

def expressions:
• Create a new function
• Bind a name to it

>>> def <name>(<formal parameters>):
    return <return expression>

Calling User-Defined Functions

from operator import mul
def square(x):
    return mul(x, x)

square(-2)

• Bind formal parameters
• Eval return expression

Environments & Values
Expressions

The environment created for the function body

The existing environment in which the call expression is evaluated
Calling User-Defined Functions

```
from operator import mul

def square(x):
    return mul(x, x)

square(-2)
```

```
mul(a,b):
    return mul(a, b)

square:
    return mul(x, x)

x: -2
square:
    return mul(x, x)

square(-2)
```

```
mul(a,b):
    return mul(a, b)

square:
    return mul(x, x)

square:
    return mul(x, x)

x: -2
square:
    return mul(x, x)

square(-2)
```

```
mul(a,b):
    return mul(a, b)

square:
    return mul(x, x)

square:
    return mul(x, x)

x: -2
square:
    return mul(x, x)

square(-2)
```

```
mul(a,b):
    return mul(a, b)

square:
    return mul(x, x)

square:
    return mul(x, x)

x: -2
square:
    return mul(x, x)

square(-2)
```

```
mul(a,b):
    return mul(a, b)

square:
    return mul(x, x)

square:
    return mul(x, x)

x: -2
square:
    return mul(x, x)

square(-2)
```

```
mul(a,b):
    return mul(a, b)

square:
    return mul(x, x)

square:
    return mul(x, x)

x: -2
square:
    return mul(x, x)

square(-2)
```

```
mul(a,b):
    return mul(a, b)

square:
    return mul(x, x)

square:
    return mul(x, x)

x: -2
square:
    return mul(x, x)

square(-2)
```

```
mul(a,b):
    return mul(a, b)

square:
    return mul(x, x)

square:
    return mul(x, x)

x: -2
square:
    return mul(x, x)

square(-2)
```

```
mul(a,b):
    return mul(a, b)

square:
    return mul(x, x)

square:
    return mul(x, x)

x: -2
square:
    return mul(x, x)

square(-2)
```

```
mul(a,b):
    return mul(a, b)

square:
    return mul(x, x)

square:
    return mul(x, x)

x: -2
square:
    return mul(x, x)

square(-2)
```

```
mul(a,b):
    return mul(a, b)

square:
    return mul(x, x)

square:
    return mul(x, x)

x: -2
square:
    return mul(x, x)

square(-2)
```

```
mul(a,b):
    return mul(a, b)

square:
    return mul(x, x)

square:
    return mul(x, x)

x: -2
square:
    return mul(x, x)

square(-2)
```

```
mul(a,b):
    return mul(a, b)

square:
    return mul(x, x)

square:
    return mul(x, x)

x: -2
square:
    return mul(x, x)

square(-2)
```

```
mul(a,b):
    return mul(a, b)

square:
    return mul(x, x)

square:
    return mul(x, x)

x: -2
square:
    return mul(x, x)

square(-2)
```

```
mul(a,b):
    return mul(a, b)

square:
    return mul(x, x)

square:
    return mul(x, x)

x: -2
square:
    return mul(x, x)

square(-2)
```

```
mul(a,b):
    return mul(a, b)

square:
    return mul(x, x)

square:
    return mul(x, x)

x: -2
square:
    return mul(x, x)

square(-2)
```

```
mul(a,b):
    return mul(a, b)

square:
    return mul(x, x)

square:
    return mul(x, x)

x: -2
square:
    return mul(x, x)

square(-2)
```

```
mul(a,b):
    return mul(a, b)

square:
    return mul(x, x)

square:
    return mul(x, x)

x: -2
square:
    return mul(x, x)

square(-2)
```

```
mul(a,b):
    return mul(a, b)

square:
    return mul(x, x)

square:
    return mul(x, x)

x: -2
square:
    return mul(x, x)

square(-2)
```

```
mul(a,b):
    return mul(a, b)

square:
    return mul(x, x)

square:
    return mul(x, x)

x: -2
square:
    return mul(x, x)

square(-2)
```

```
mul(a,b):
    return mul(a, b)

square:
    return mul(x, x)

square:
    return mul(x, x)

x: -2
square:
    return mul(x, x)

square(-2)
```

```
mul(a,b):
    return mul(a, b)

square:
    return mul(x, x)

square:
    return mul(x, x)

x: -2
square:
    return mul(x, x)

square(-2)
```

```
mul(a,b):
    return mul(a, b)

square:
    return mul(x, x)

square:
    return mul(x, x)

x: -2
square:
    return mul(x, x)

square(-2)
```

```
mul(a,b):
    return mul(a, b)

square:
    return mul(x, x)

square:
    return mul(x, x)

x: -2
square:
    return mul(x, x)

square(-2)
```
Example: Function Application

```python
def square(x):
    return mul(x, x)

def sum_squares(x, y):
    return add(square(x), square(y))

sum_squares(5, 12)
```

```
25
```

```
169
```

```python
square(5):
    mul(5, 5)

square(12):
    mul(12, 12)
```

```
144
```