Imperative Languages: Python

Functional programming vs Imperative

eg sml vs python

functional programming is a programming paradigm where programs are constructed by applying and composing functions. It is a declarative programming paradigm in which function definitions are recursively defined as trees of expressions that each return a value

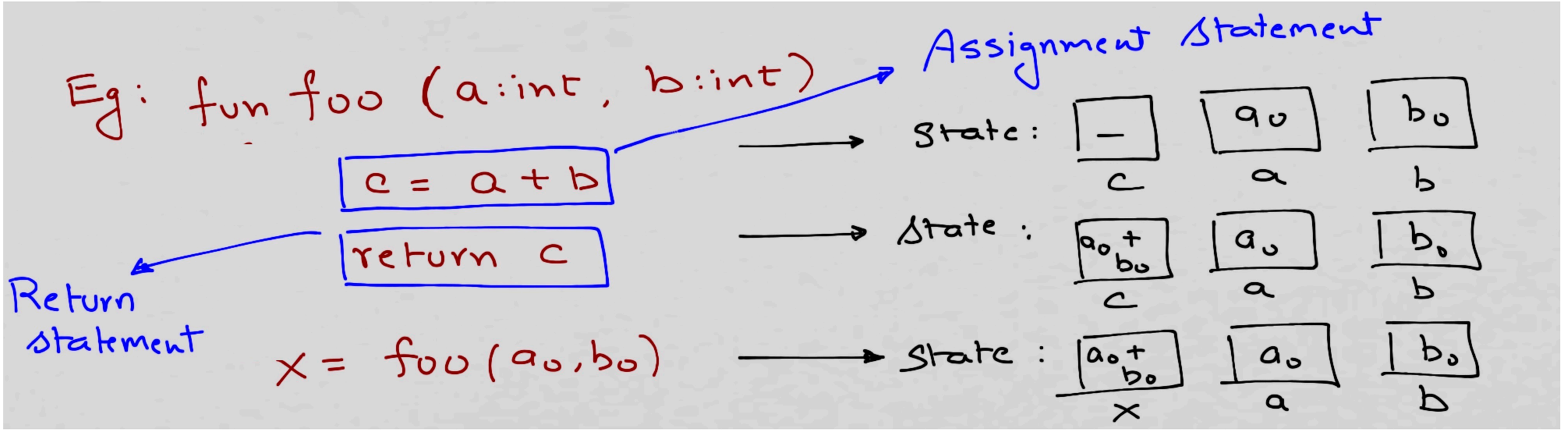
functional languages — sml, lisp, scheme, Swift, Haskell, scala …

Imperative programming is a programming paradigm where programs are constructed by applying a sequence of imperative statements which change the state of the program. an imperative program consists of commands for the computer to perform. Imperative programming focuses on describing *how* a program operates. The term is often used in contrast to declarative programming, which focuses on *what* the program should accomplish without specifying *how* the program should achieve the result.

imperative languages — fortran, C, C++, python, java …..

Imperative model of computation

- state is the key - gets modified by the steps of computation
- steps of computations — via commands or instructions called **statements**



Imperative computation

- Each statement has a precondition and a postcondition
  (properties of the states just before and after the statement)

Eg:

(SWAP : Precondition $(a == a_0 \wedge b == b_0)$

```
temp = a
a = b
b = temp
```

(PostCondition: $(a == b_0 \wedge b == a_0)$

Statements (contd)

- assert statement — *a logical statement that evaluates a condition at a particular step of the computation - a logical statement used to document your correct design*
  - if then else

```
if ( boolean expression ):
    STATEMENT
    STATEMENT
else:
    STATEMENT
    STATEMENT
```

  - a BLOCK is made up of statements that are at the same indentation level.
  - You can NOT mix indentation levels within the same block!
  - indented block of code following an if statement is executed if the boolean expression is true, otherwise it is skipped.

Python Common Operators: +, -, *, /, <, >, <=, >=, ==, %

- Some operators should be familiar such as Addition (+), Subtraction (-), Multiplication (*), and Division (/).
- comparison operators, such as Less-Than (<), Greater-Than (>), Less-Than-or-Equal(<=), Greater-Than-or-Equal (>=), and **Equality-Test (==)**. These operators produce a True or False value.
- examples, along with the new value they produce when evaluated:

```
10 > 5                    produces True
10 < 5                    produces False
10 / 3.5                  produces 2.8571428571
10 / 3                    produces 3   like div
10 % 3                    produces 1   mod
"Hi" + " " + "Jay!"       produces "Hi Jay!"
```

Warn! Operator Overloading!

- NOTE! Some operators will work in a different way depending upon what their operands are.
- eg if you “add” two or more strings, the + operator produces a concatenated version of the strings: `“Hi” + “Jay”` produces “HiJay”
- Multiplying strings by a number repeats the string!

`“Hi Jay” * 3` produces “Hi JayHi JayHiJay”

Variables and Data Types

- Variables are names that can point to data.
- They are useful for saving intermediate results and keeping data organized.
- The assignment operator (=) assigns data to variables.
- like sml all data has an associated data Type. You can find the “Type” of any piece of data by using the type() function:

  `type( “Hi!”)` produces <type 'str'>

  `type( True )` produces <type 'bool'>

  `type( 5)` produces <type 'int'>

  `type(5.0)` produces <type 'float'>
- Note Unlike sml python variables are dynamically typed

Program Example

Find the area of a circle given the radius:

```
def findArea( Radius ):
    pi = 3.1459
    area = pi * Radius * Radius
    return area

area = findArea(10)
print “area for circle of radius”,10,”is”,area
```

will print ‘area for circle of radius 10 is 314.59’ to the screen.

Comparing SML & Python Implementation

factorial(n)

- recall sml

```
fun factorial (n) =
    if n = 0 then 1
    else n * factorial (n-1);
```

- python equivalent

```
def factorial(n):
    if n==0:
        return 1
    else:
        return n * factorial(n-1)
```

we can use recursive (functional) style almost identically

Comparing SML & Python Implementation

power(x,n)

- recall sml

```
fun power(x, n) =
   if n = 0   then 1.0
   else x * power(x, n - 1);
```

- python equivalent

```
def power(x,n):
    if n==0:
        return 1.0
    else:
        return x * power(x,n-1)
```

Comparing SML & Python Implementation

sum(foo,a,b)

- recall sml

```
fun sum (foo,a,b) =
    if a > b then 0
    else foo(a) + sum (foo, a+1, b);
```

- python equivalent

```
def sum(foo,a,b):
    if a>b:
        return 0
    else:
        return foo(a) + sum(foo,a+1,b)
```

We can pass functions as parameters. Indeed python also has lamda, map, filter, reduce

Looping, a better form of repetition.

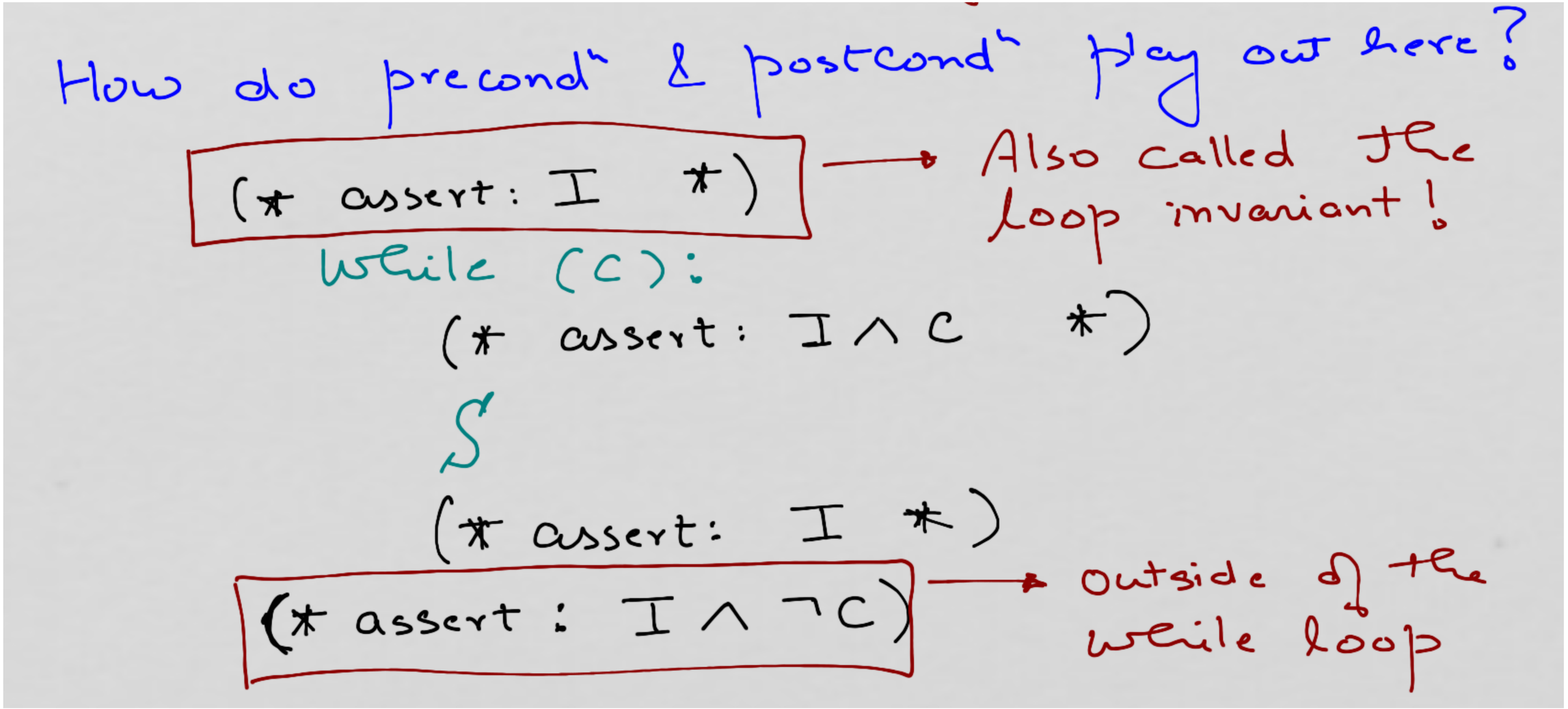
- If you want to repeat a computation many times,we can use a loop to make the computer do the work for us.
- One type of loop is the “while” loop. The while loop repeats a block of code until a boolean expression is no longer true.

```
while (C) :
    S
```

```
while condition C is true keep executing
  the Block of statements S
```

Loops and Invariants

Loops closely related to iteration that we have already studied



Iterative factorial with while loop

Eg:

SML

```
fact_iter (n,f,c) =
    if (c = n) then f
    else fact_iter (n, f * (c+1), c+1)
```

Imperative code

```
fact_iter (n) :
        f = 1
        c = 0
    (* I:
    while ( c != n) :
            c = c + 1
            f = f * c
    (* I: f = c! ∧
            0 ≤ c ≤ n *)
(* f = c! ∧ 0 ≤ c ≤ n ∧
        c == n *)
```