**Who** writes all of the assembly language that gets executed?

### Python

**How does Python function?**

3 Hmm problems due Mon. 3/9

---

**Hmmm**

RAM
registers
1-bit memory: flip-flops arithmetic
bitwise functions
logic gates
transistors / switches

---

**CS 5 this week**

---

**Functions vs. instructions**

- **def main():**
  - `x = input()`
  - `y = fun(x)`
  - `print y`

- **def fun( x ):**
  - `y = x*(x-1)`
  - `return y`

**Fun: Python**

---

**Instructions: Hmmm**

- `read r1` // input
- `call r14 4` // function call
- `write r13` // output
- `halt`
- `copy r13 r1`
- `addn r1 -1`
- `mul r13 r13 r1`
- `jumpr r14` // "return"
- `let's agree to put the function input into r1`
- `r1`
- `let's agree to put the return line # into r14`
- `r14`
- `let's agree to put the return value in r13`
- `r13`
- `def main():`
  - `r1 = input()`
  - `r13 = fun(r1)`
  - `print r13`
- `def fun( r1 ):`
  - `r13 = r1*(r1-1)`
  - `return r13`

---

**Automatic assembly of functions...**

---

**conventions!**
functions \neq \text{instructions}

\text{calln} \quad \text{(call)}

\text{jump} \quad \text{(return)}

\text{read} \ r1 \ 0 \ 1 \ 2 \ 3
\text{calln} \ r14 \ 4
\text{write} \ r13
\text{halt}
\text{copy} \ r13 \ r1
\text{addn} \ r1 \ -1
\text{mul} \ r13 \ r13 \ r1
\text{jump} \ r14

\text{Instructions: Hmmm}

\text{Input value: x}
\text{Final result - ret. value - in progress}
\text{return address register}

\text{r1}
\text{r13}
\text{r14}

\text{Input value: x}
\text{Final result - ret. value - in progress}
\text{return address register}

\text{r1}
\text{r13}
\text{r14}

\text{We'd need the earlier value of x} (r1) \ldots \text{after the function call!}

\text{There's no stopping the destruction... let's just keep it all in one place!}

\text{Problem}
\text{Functions clobber stuff!}

\text{Solution}
\text{CALL our function and let it clobber stuff!}

\text{College administrations like this approach, too...!}

\text{We want it to SEEM like each function has its own registers!}

\text{... but we want LOTS of function calls.}

\text{Functions will overwrite registers}

\text{def main():}
\quad r1 = \text{input}()
\quad r13 = \text{fun}(r1)
\quad \text{print} \ r13

\text{def fun( r1 ):}
\quad r13 = r1 \ast (r1-1)
\quad \text{return} \ r13
### Python

```python
x = input()  # Input value: x

def fac(x):
    if x == 0:
        return 1
    else:
        RES = fac(x - 1)
        return x * RES

y = fac(x)  # y = fac(x)

print(y)
```

### Hmmm

```
00 read r1
01 setn r14 42
02 nop
03 calln r14 7
04 nop
05 write r13
06 halt

07 jnezn r1 11
08 setn r13 1
09 nop
10 jump r14
11 pushr r1 r15
12 pushr r14 r15
13 addn r1 -1
14 nop
15 calln r14 7
16 nop
17 popr r14 r15
18 popr r1 r15
19 mul r13 r1 r13
20 jump r14
```

### factorial function

```
def fac(x):
    if x == 0:
        return 1
    else:
        RES = fac(x - 1)
        return x * RES
```

### factorial function (Base Case)

```
def fac(x):
    if x == 0:
        return 1
```

### factorial function (Recursive Call)

```
def fac(x):
    if x == 0:
        return 1
    else:
        RES = fac(x - 1)
        return x * RES
```
For an input of 0, trace what happens here...

For an input of 3, trace what happens here...

You try it

Start with this recursive factorial function...

\textbf{fac}( N )

Strategy: \textbf{standardize registers}

Simplify by having a \textbf{standard place} for \textbf{standard data}.

How much will you need to \textbf{change} to implement a recursive power function? \textbf{power}( B, N )

Programmatically!

Organizationally
def fac(x):
    """ factorial w/ printing """
    print("x is", x)
    if x == 0:
        print("x:", x, " Res: 1")
        return 1
    else:
        print("Next: fac(" , x-1, ")")
        smaller = fac( x-1 )
        result = x * smaller
        print("x:",x," Res:",result)
    return result