Instruction Cycle

1. An instruction is fetched from the memory address stored in the program counter (PC) and is stored in the instruction register (IR). The PC is incremented.

2. The decoder interprets the instruction. If it is a jump, the PC is reset and the cycle ends. Otherwise, any required data is fetched from main memory and is placed in data registers.

3. The CPU executes the instruction by reading values from registers, performing arithmetic or logical functions on them, and writing the result into a register.

4. The result is stored in main memory or is sent to an output device.

5. Interrupts are handled.
Python Control Structures

Python control structures translate into sequences of machine instructions.

FOR loop
- The body translates into arithmetic and logical instructions.
- The loop variable is initialized before the first instruction and is updated after the last instruction.
- A conditional jump returns control to the first instruction when the termination condition is false.

WHILE loops are similar.

Exceptions are raw jumps.

Jumps are dangerous because they can transfer control anywhere.
Call Stack

- Function calls are coordinated using a *call stack*.
- Each function call generates a *stack frame* that stores its arguments, local variables, space for return values, and a return address.
- When $f$ calls $g$, a $g$-frame is initialized and control is transferred to the first instruction of $g$.
- When $g$ returns, execution of $f$ resumes at the $g$ return address.
- Nested function calls are handled by processing stack frames in first-in first-out order (FIFO).
Call Stack

drawline

- arguments
- local variables
- return value
- return address

drawbox

- arguments
- local variables
- return value
- return address
Call by value: parameters and argument are different memory addresses.

Call by reference: parameters and argument are same memory address.

Call by value is simpler and safer.

Call by reference is more efficient.

Python uses call by value for numbers and strings, but call by reference for lists and objects.
Recursive Functions

A recursive function contains calls to itself in its body.
These calls are regular function calls.
The programmer must ensure that the calls terminate.
Example: compute $1 + 2 + \cdots + n$.

```python
def sum(n):
    if n == 1:
        return 1
    return n + sum(n - 1)
```

Base case of $n = 1$ has no recursion.
Recursive calls terminate because $n$ decreases.
Call Stack

- sum(1)
  - local variable: \( n = 1 \)
  - return value: 1
  - return address: \(<\text{sum}(2)\>\)

- sum(2)
  - local variable: \( n = 2 \)
  - return value: \( 2 + \text{sum}(1) \)
  - return address: \(<\text{sum}(3)\>\)

- sum(3)
  - local variable: \( n = 3 \)
  - return value: \( 3 + \text{sum}(2) \)
  - return address: \(<\text{sum}(4)\>\)

- sum(4)
  - local variable: \( n = 4 \)
  - return value: \( 4 + \text{sum}(3) \)
  - return address: \(<\text{main} >\)
Recursive Palindrome

def palindrome(s):
    if len(s) < 2:
        return True
    if s[0] != s[-1]:
        return False
    return palindrome(s[1:-1])

- Base case 1: empty string or one character.
- Base case 2: first character differs from last.
- Recursive case: strip first and last characters.