

# What Is Python?

---



**Jason Olson**  
Staff Software Engineer

@jolson88 [www.jolson88.com](http://www.jolson88.com)



It's okay  
if you *don't* understand  
the code shown



# What Is Python?



- Syntax
- General-purpose
- Multi-paradigm
- Interpreted
- Garbage-collected
- Dynamically-typed



# Python Syntax

---



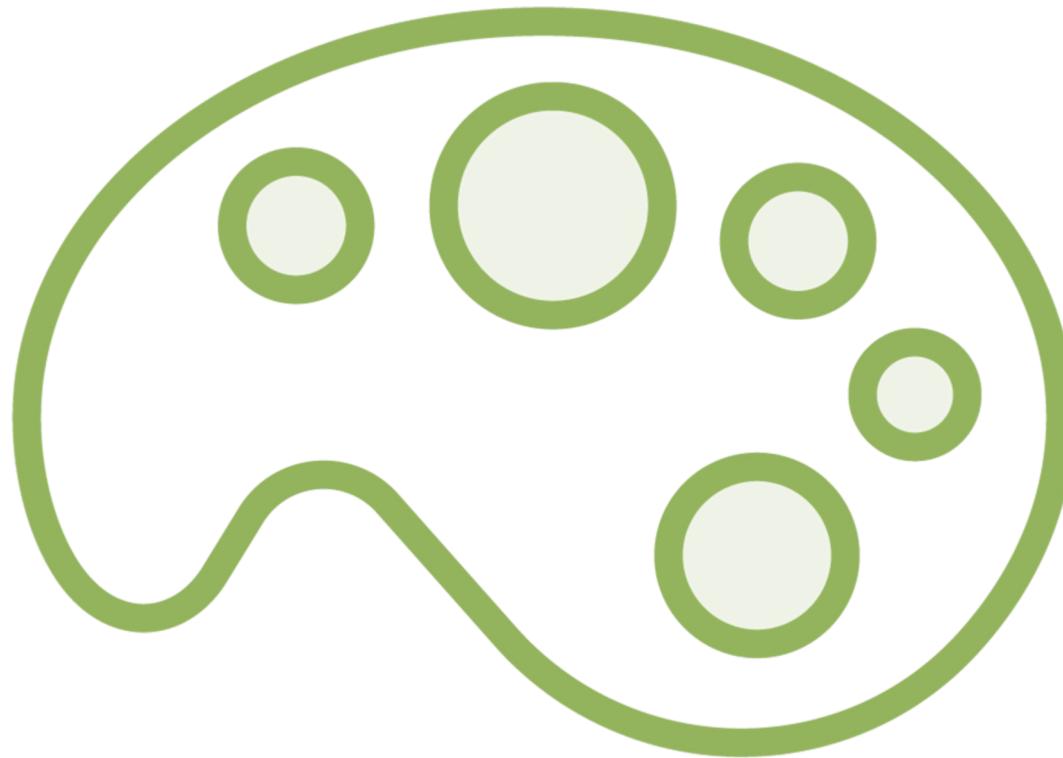
# C-like Syntax

```
long some_function(); ←  
/* int */ other_function(); ←  
  
/* int */ calling_function()  
{ →  
    long test1;  
    register /* int */ test2;  
  
    test1 = some_function(); ←  
    if (test1 > 0)  
        test2 = 0;  
    else  
        test2 = other_function();  
    return test2;  
} →
```

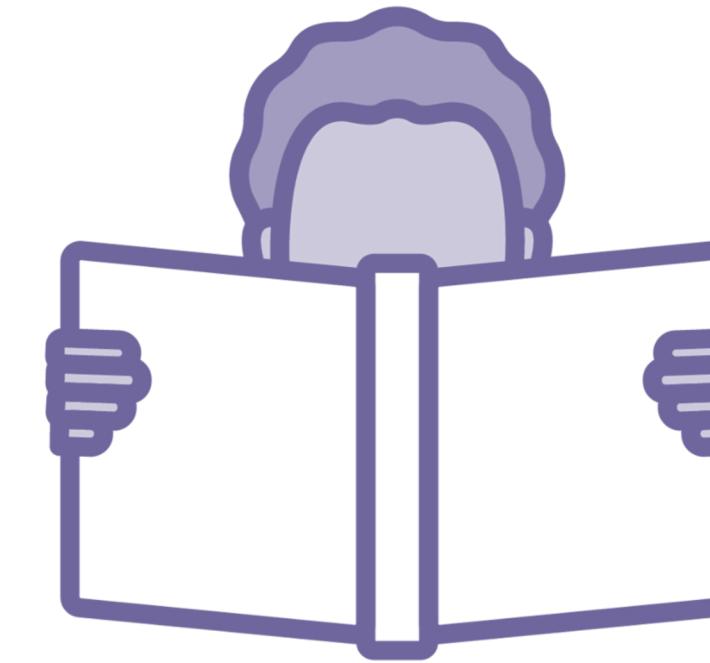
C, C++, Java, JavaScript, C#, Go, PHP, etc.



# Some Python Principles...



**Beautiful is better than ugly**



**Readability counts**



# Significant Whitespace

```
n = int(input('Type a number, and its factorial will be printed: '))

if n < 0:
    raise ValueError('You must enter a non-negative integer')

fact = 1

for i in range(2, n + 1):
    fact *= i

print(fact)
```

Easier to read and understand

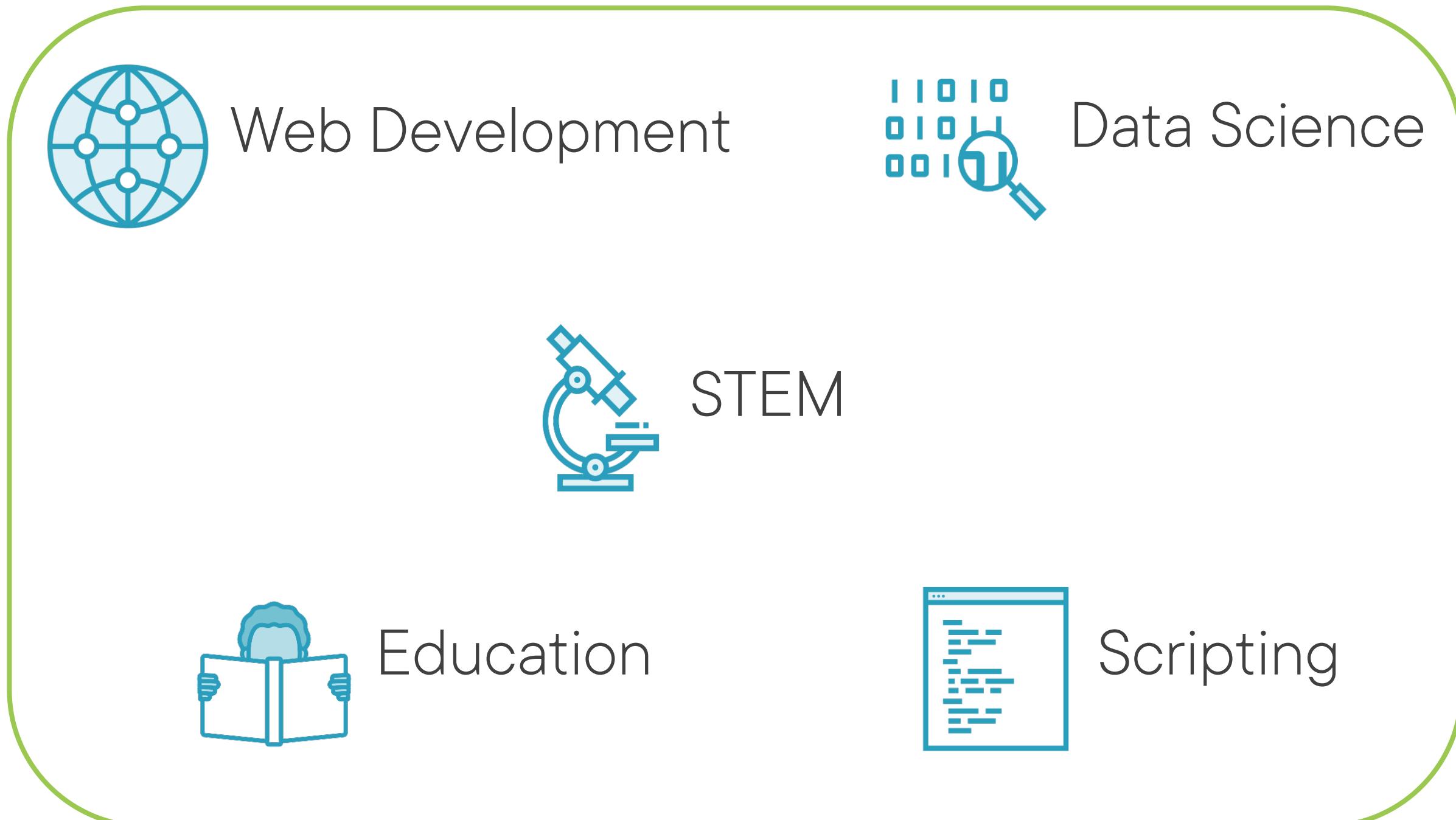


# General-purpose and High-level

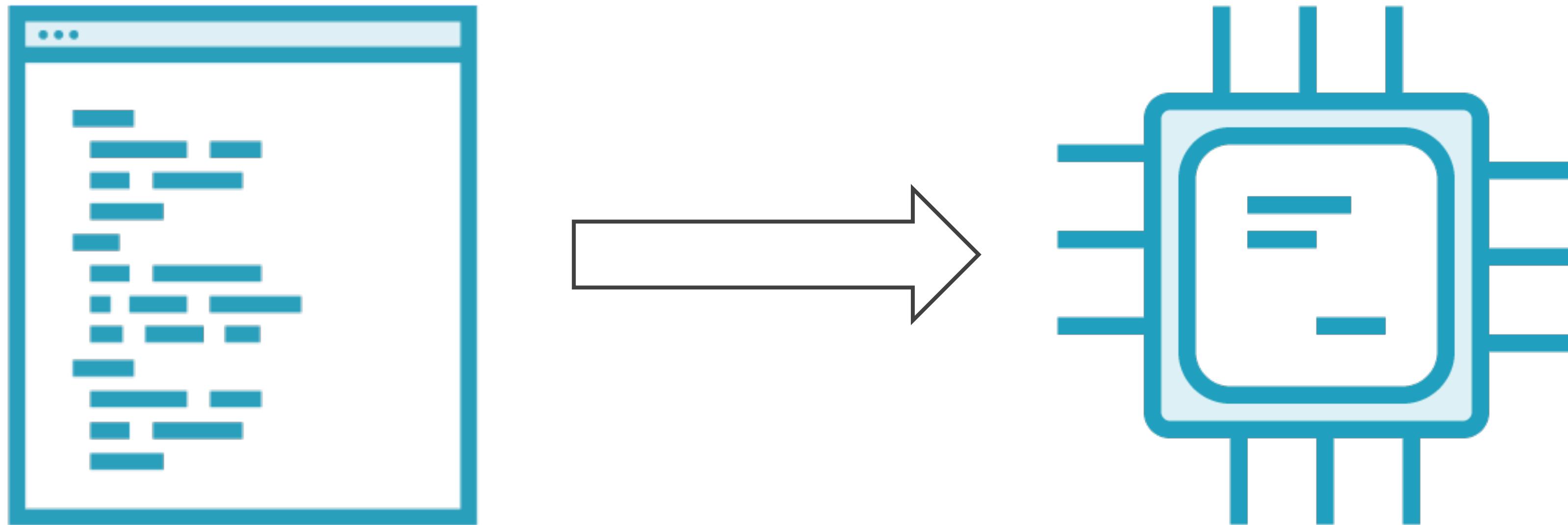
---

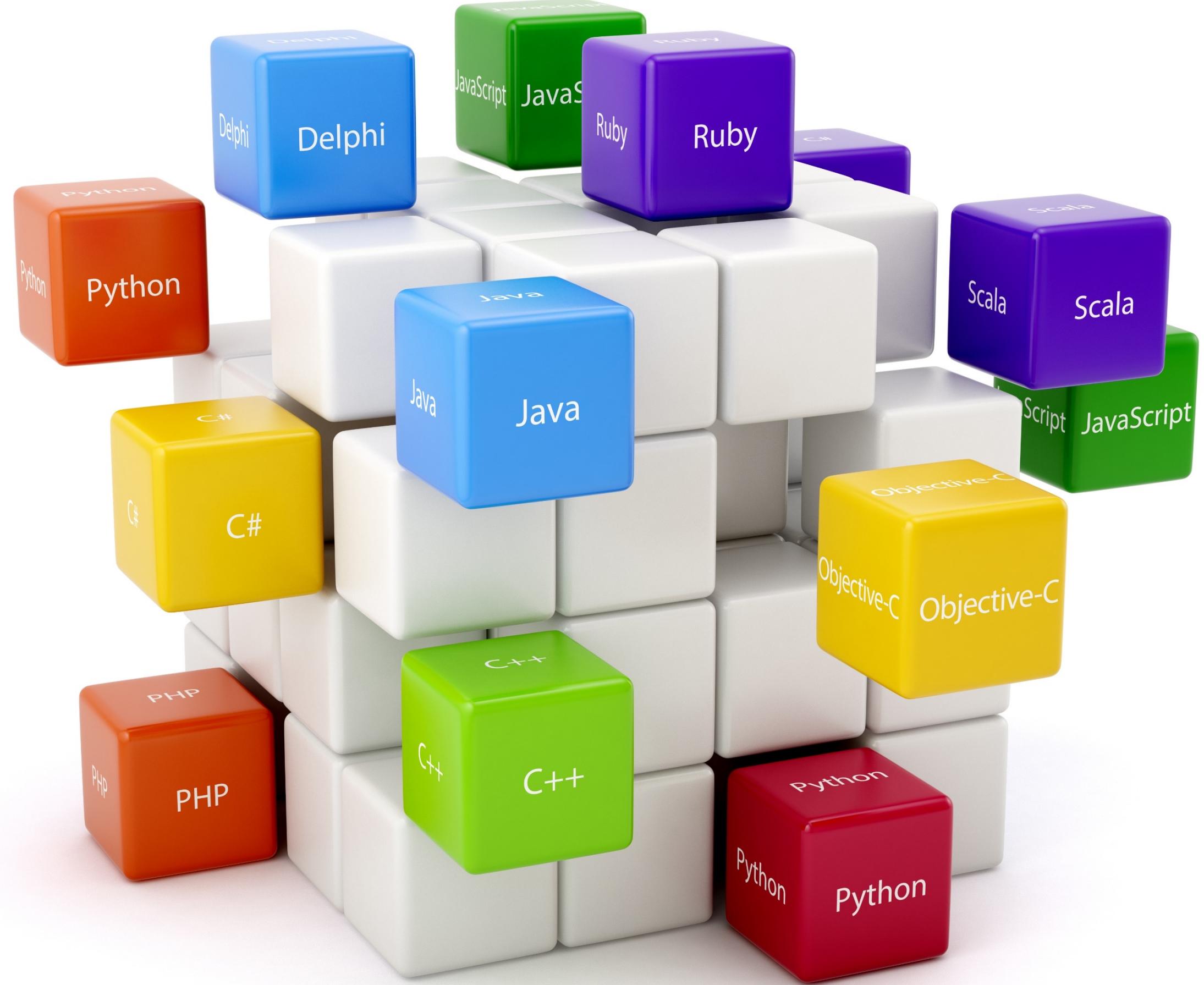


# General-purpose



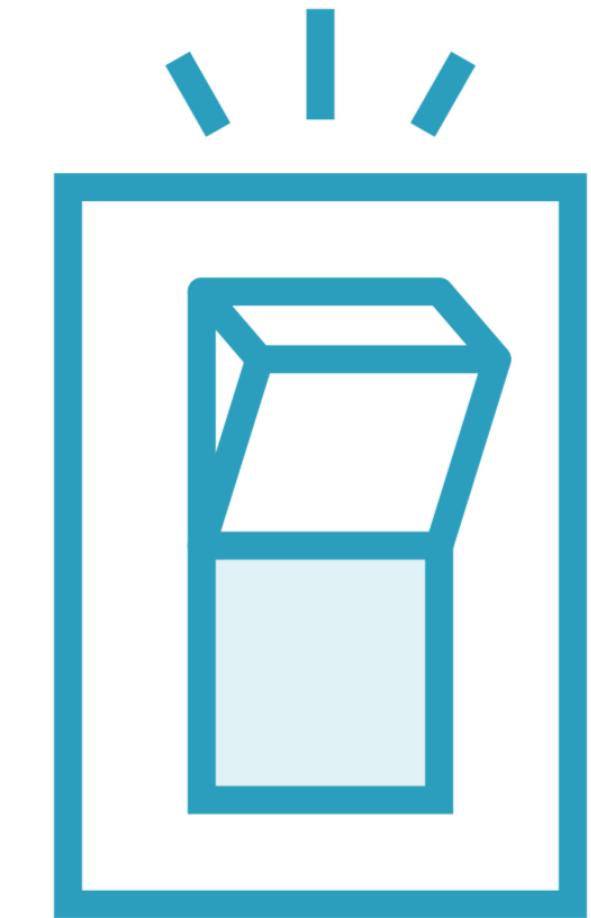
# High-level Programming Languages





# How Computers Work

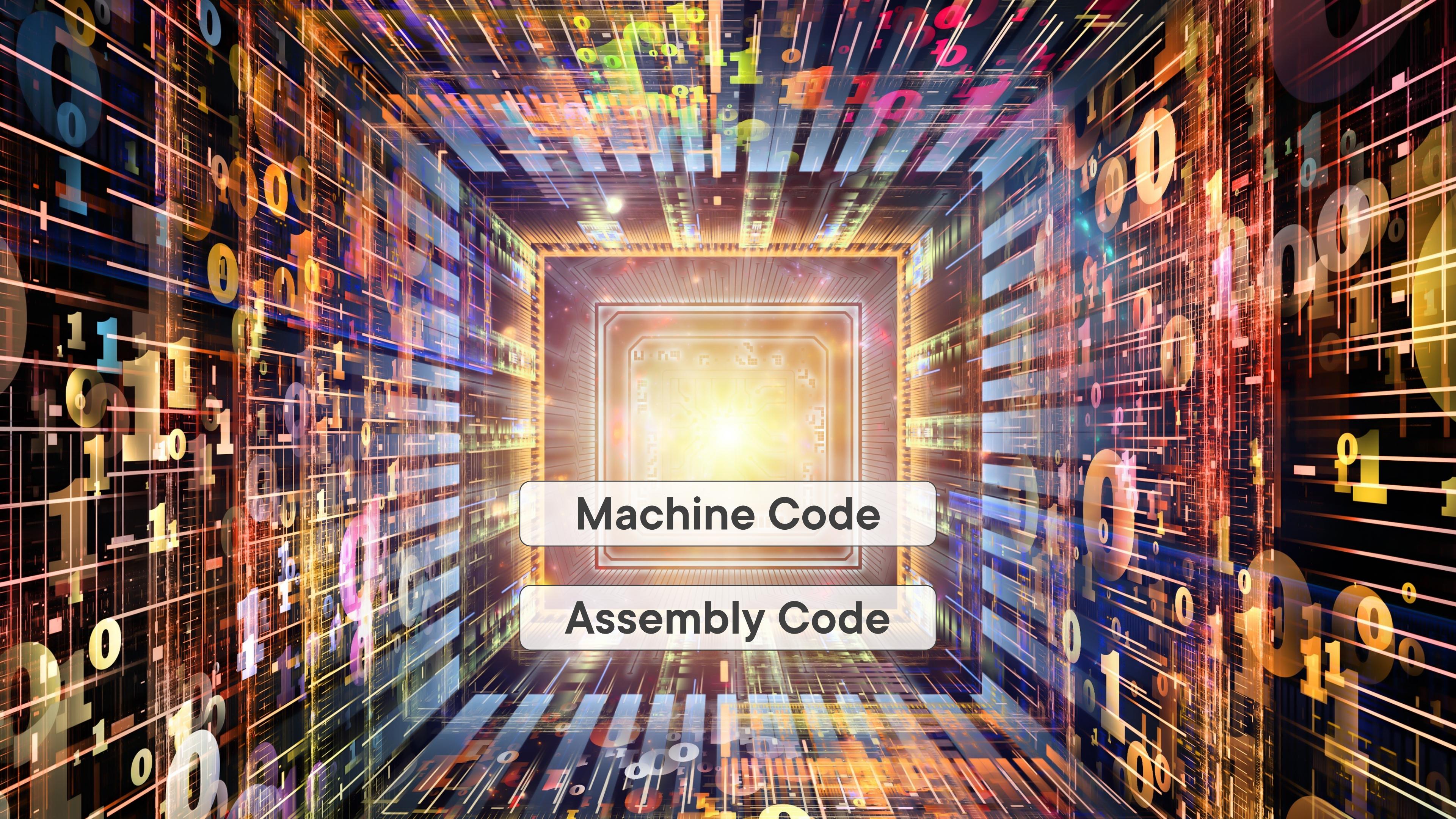
0 1 0 0



**Binary**

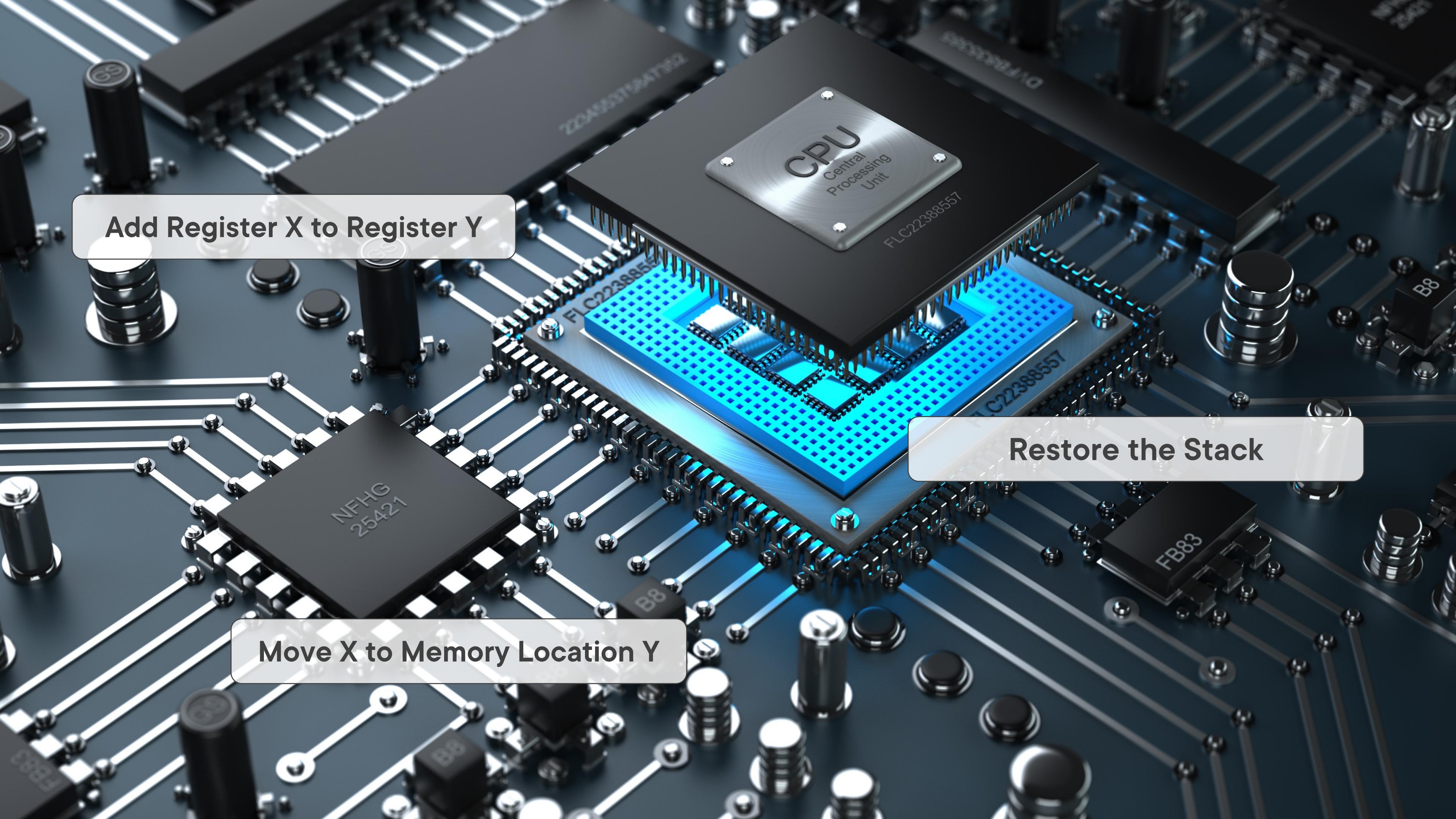






**Machine Code**

**Assembly Code**



Add Register X to Register Y

Restore the Stack

Move X to Memory Location Y

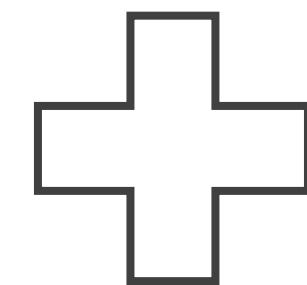


Display Image

Generate Graph

Play Video

# General-purpose



High-level

---

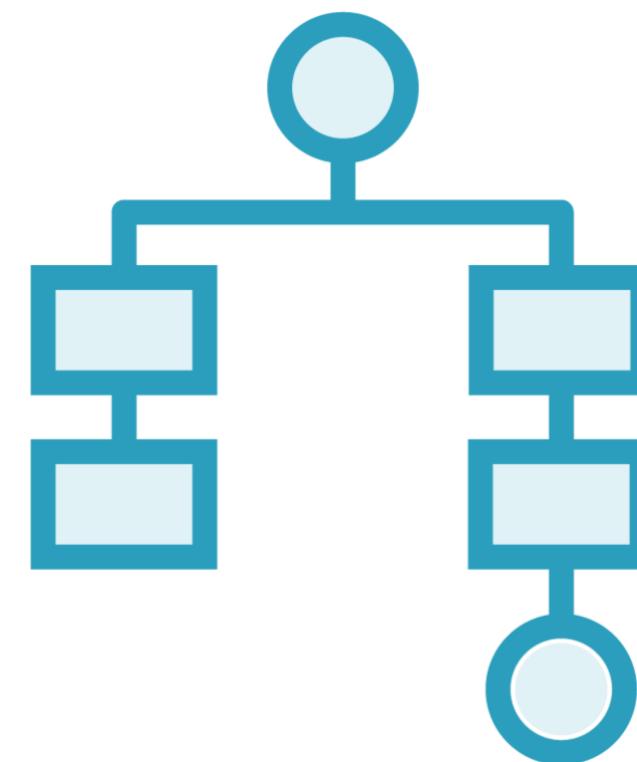


# Multi-paradigm

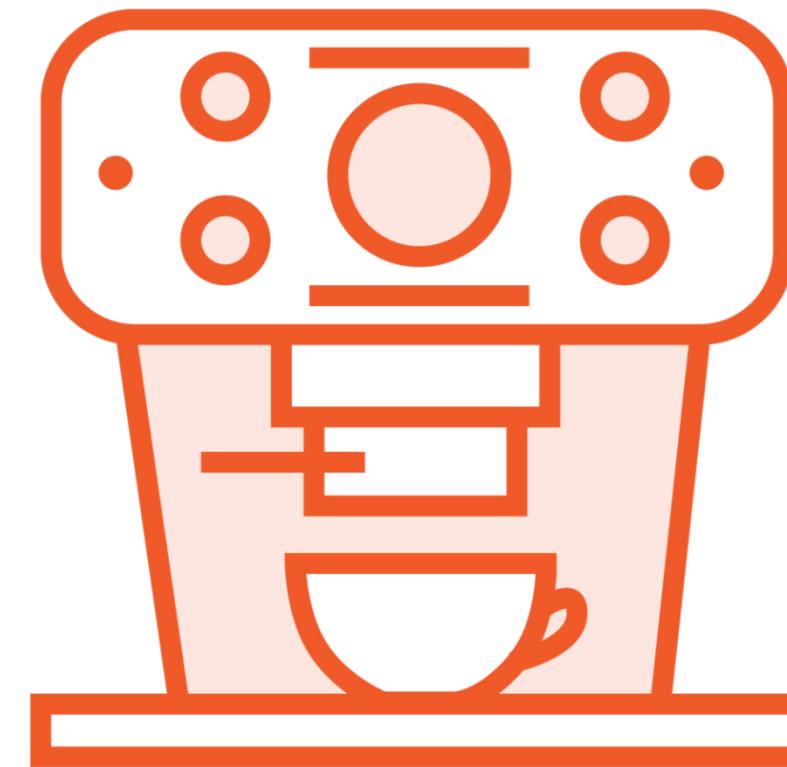
---



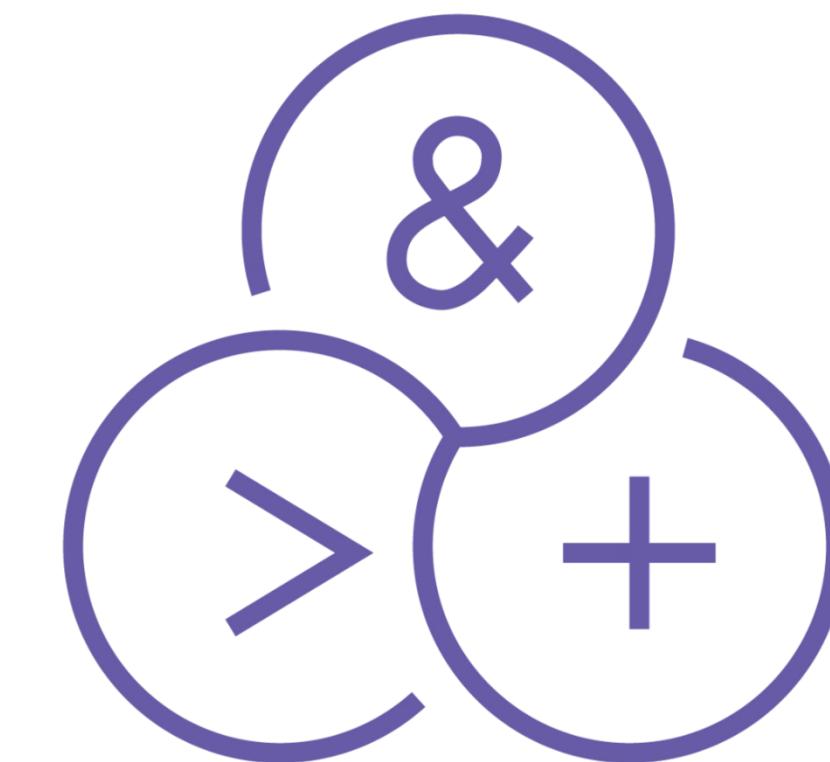
# Programming Paradigms



**Structured  
Programming**



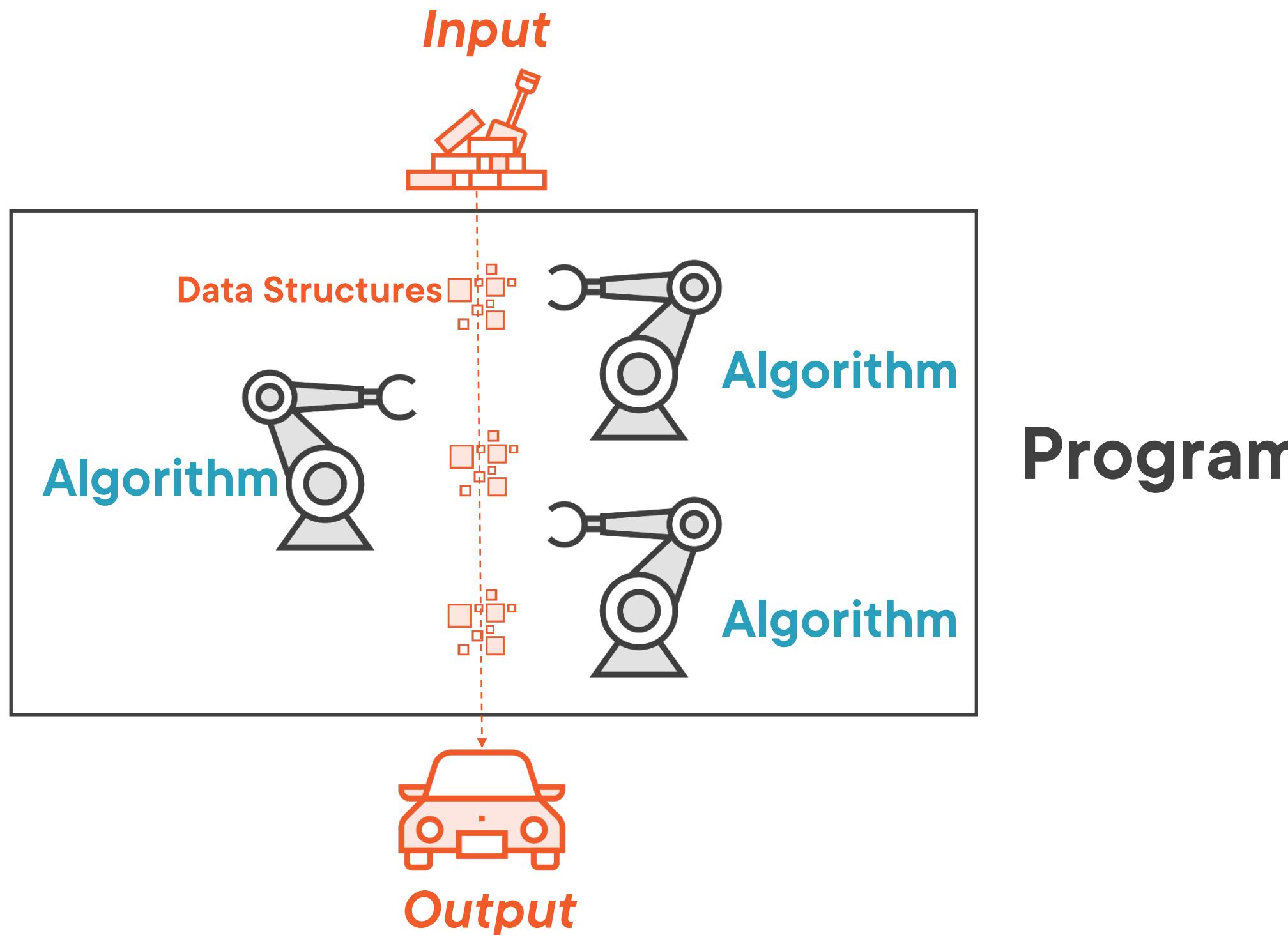
**Object-oriented  
Programming**



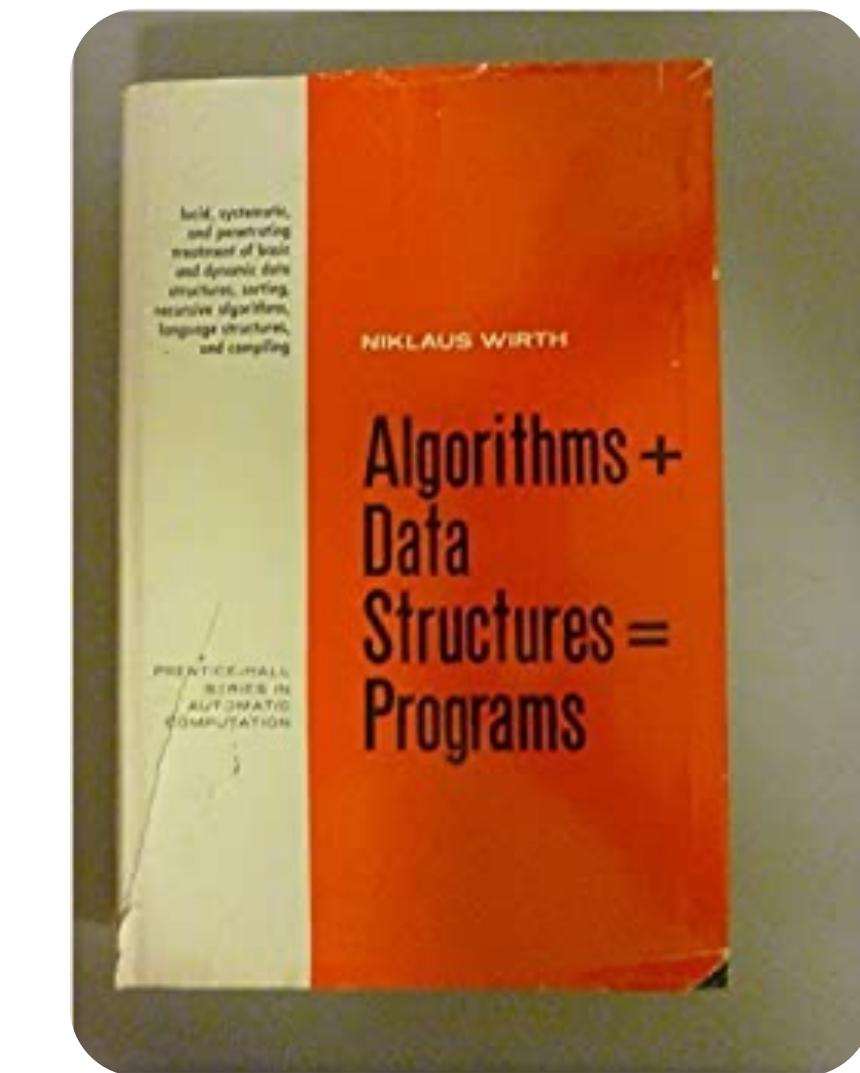
**Functional  
Programming**



# Programming 101

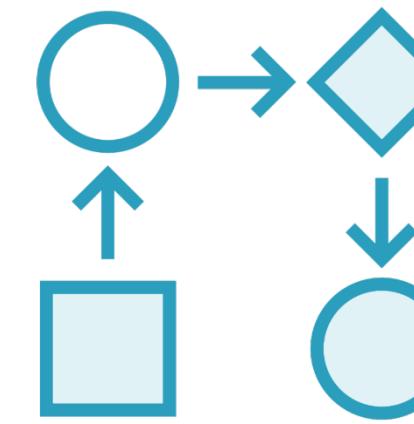


Program



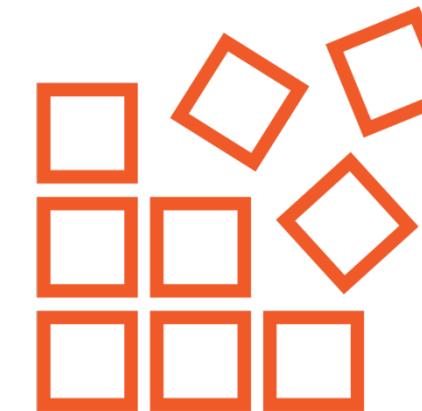
# Structured Programming

## Control Structures



**IF this THEN a ELSE b**  
**DO something UNTIL that**  
**WHILE this DO that**

## Subroutines



**Procedures**  
**Functions**  
**Methods**



```
push    ebp
mov     ebp, esp
mov     eax, [ebp+8]
mov     ecx, [ebp+0Ch]
lea     eax, [ecx+eax*2]
pop     ebp
retn
push    ebp
mov     ebp, esp
push    ecx
mov     eax, [ebp+0Ch]
mov     ecx, [eax+4]
push    ecx
call    dword ptr ds:_imp__atoi
add     esp, 4
mov     [ebp-4], eax
mov     edx, [ebp-4]
push    edx
mov     eax, [ebp+8]
push    eax
call    _sub
add     esp, 8
mov     esp, ebp
pop     ebp
retn
```



# Structured Programming in Python

```
def fizzBuzz(n):
    for i in range(1, 20):
        if i % 3 == 0 and i % 5 == 0:
            print("FizzBuzz")
        elif i % 3 == 0:
            print("Fizz")
        elif i % 5 == 0:
            print("Buzz")
        else:
            print(str(i))

fizzBuzz(20)
```



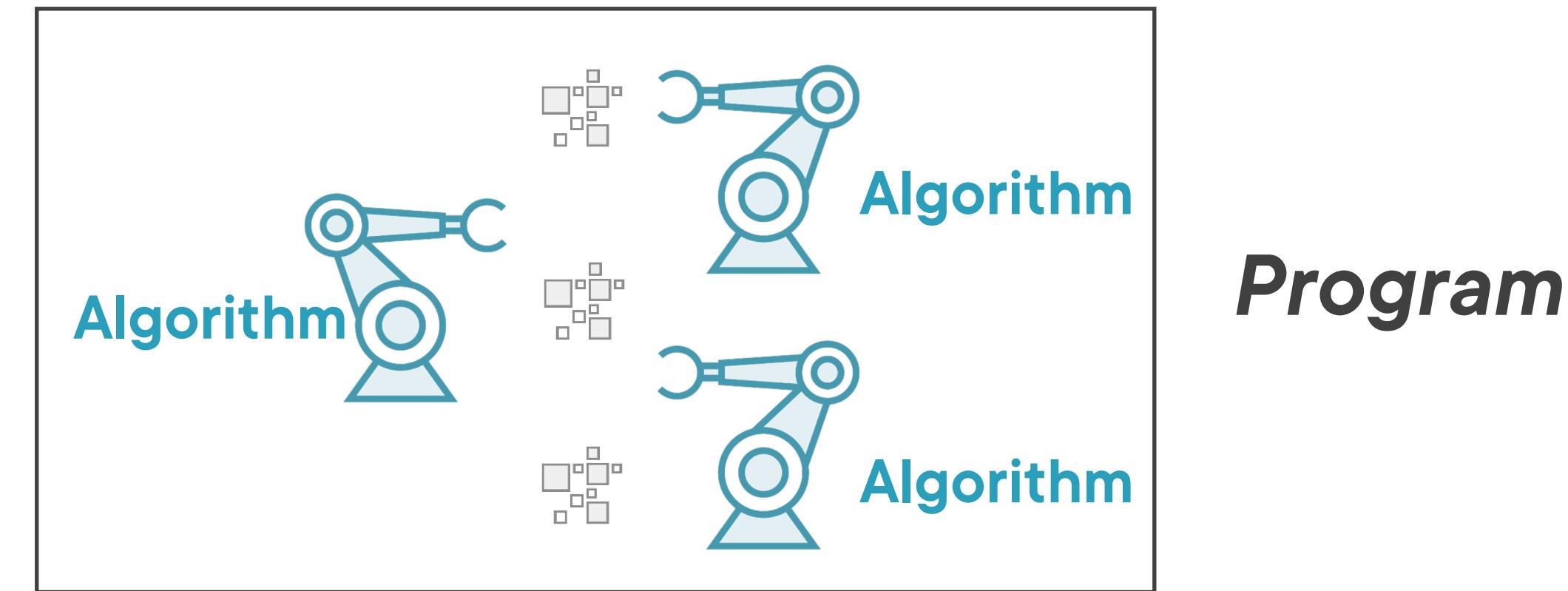
**Control Structures**



**Subroutines**



# Functional Programming



# Functional Programming

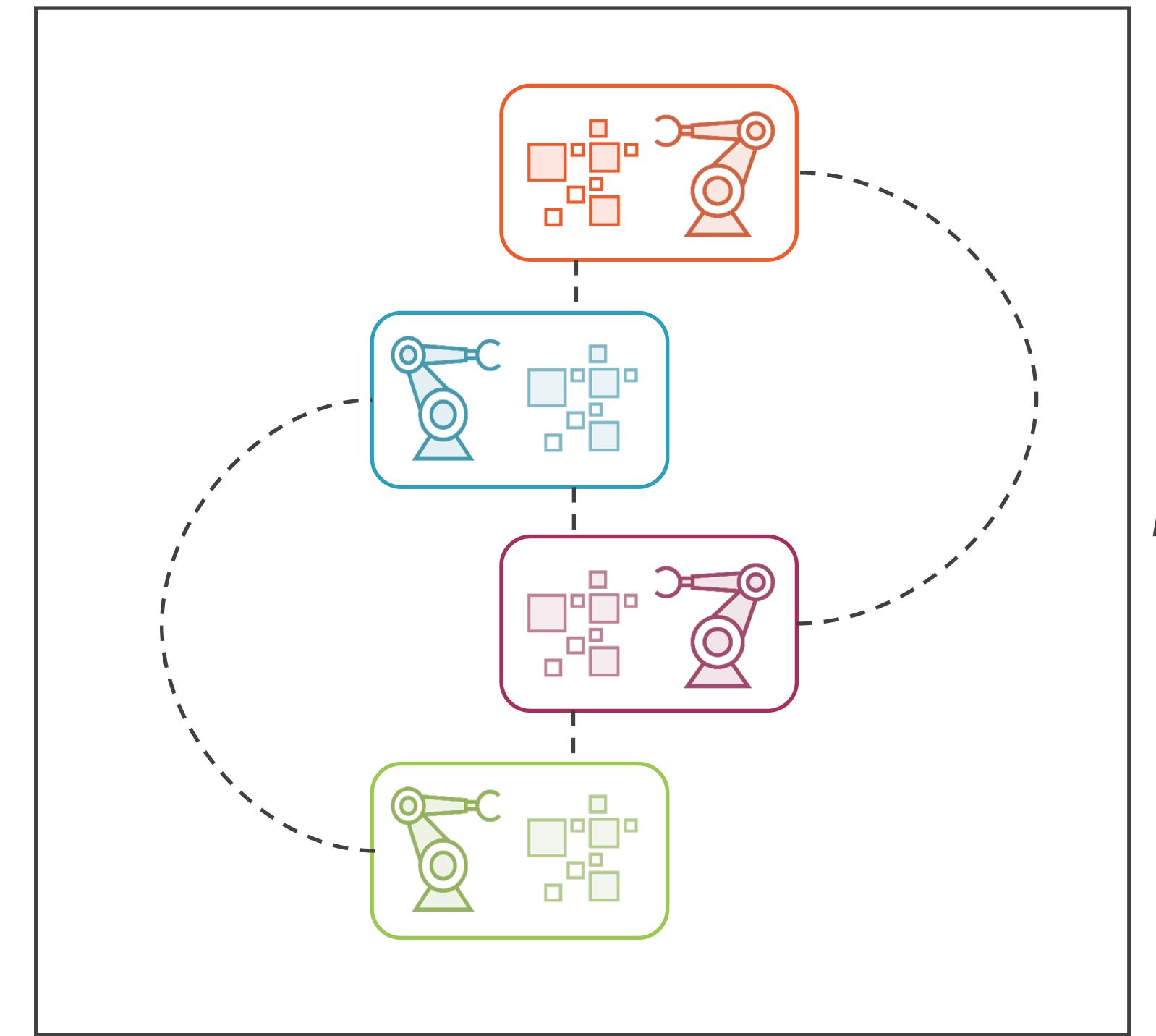


***“It is better to have 100 functions operate on one data structure than to have 10 functions operate on 10 data structures.”***

**Alan J. Perlis**  
**Computer**  
**Scientist**



# Object-oriented Programming



# Python Is Multi-paradigm

## Functional Programming

```
def reverse(s):
    return s[::-1]
animals = ["wolf", "horse", "cat",
"dog"]
iterator = map(reverse,
sorted(animals))
for i in iterator:
    print(i);
...
tac
god
esroh
flow
```

## Object-oriented Programming

```
class Animal:
    def pet(self):
        print("...")

class Dog(Animal):
    def pet(self):
        print("bark!")

class Cat(Animal):
    def pet(self):
        print("meow!")

def pet_animal(animal):
    animal.pet()

pet_animal(Dog())
pet_animal(Cat())
```



# Python has you covered!

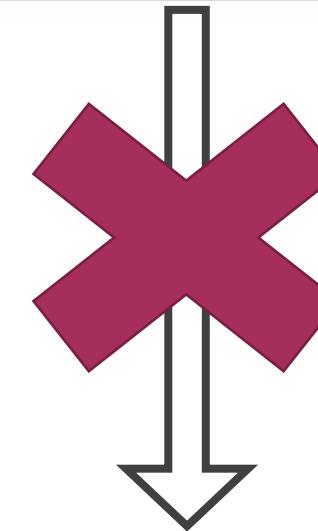


# Python Is Interpreted

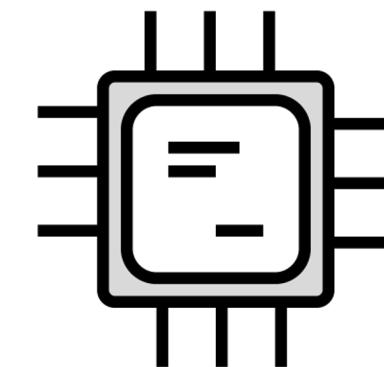
---



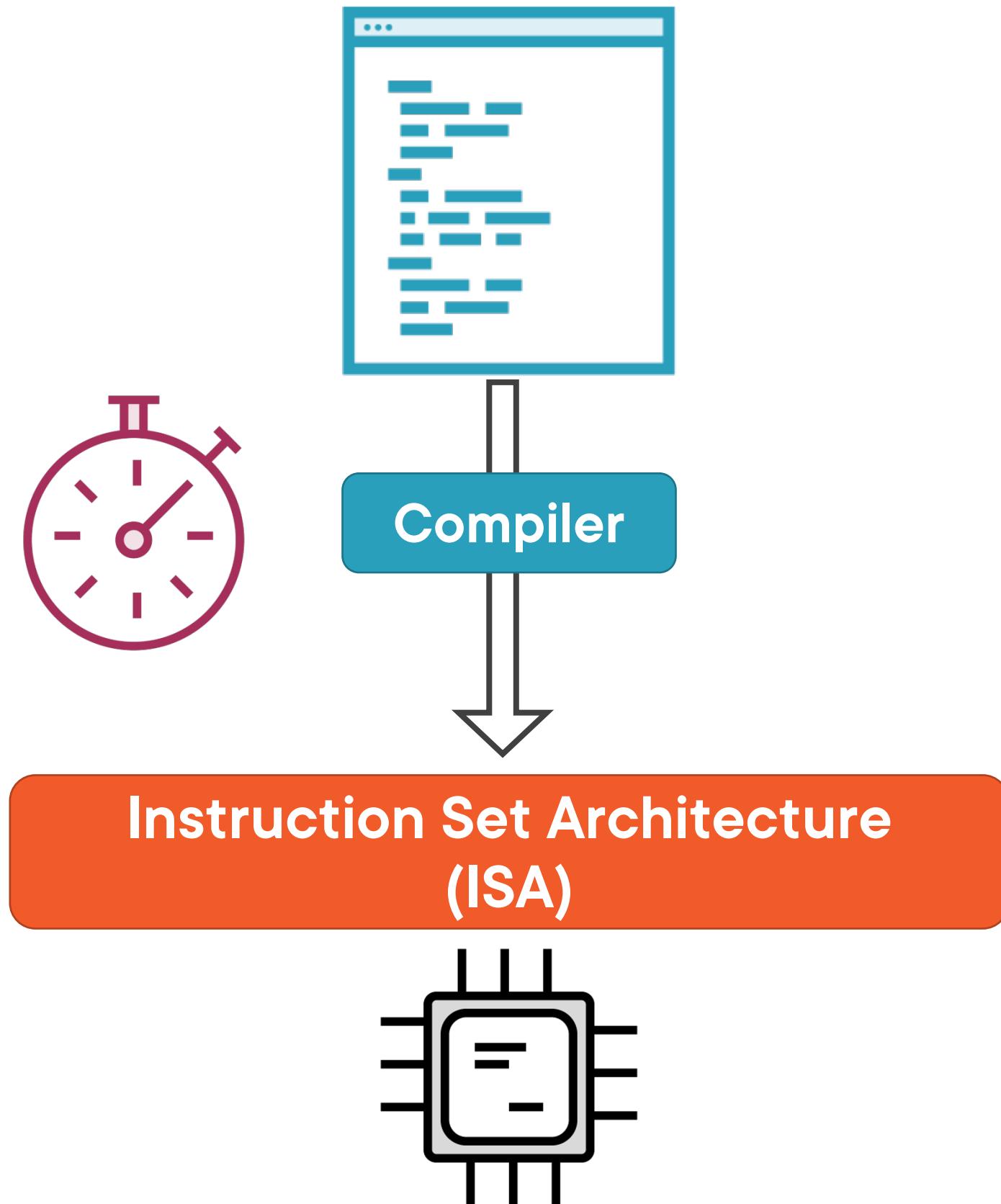
# The CPU's Language



**Instruction Set Architecture  
(ISA)**

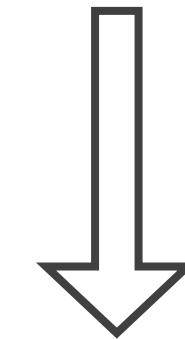
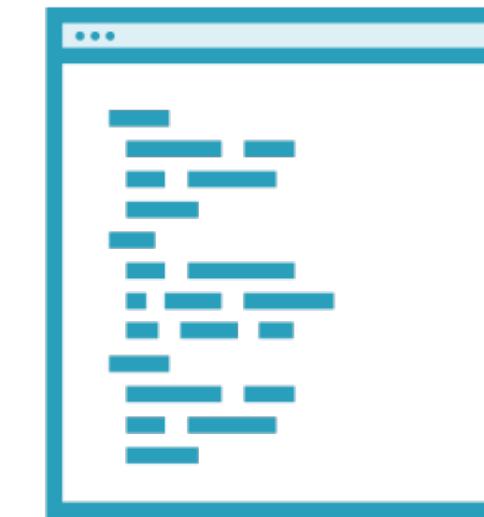


# Compiling



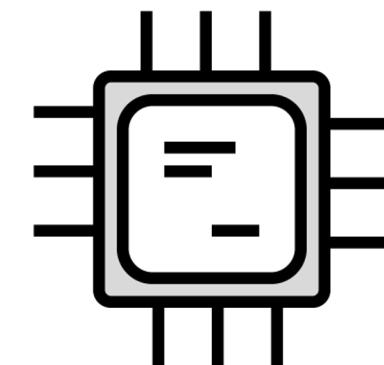


# Python Is Interpreted

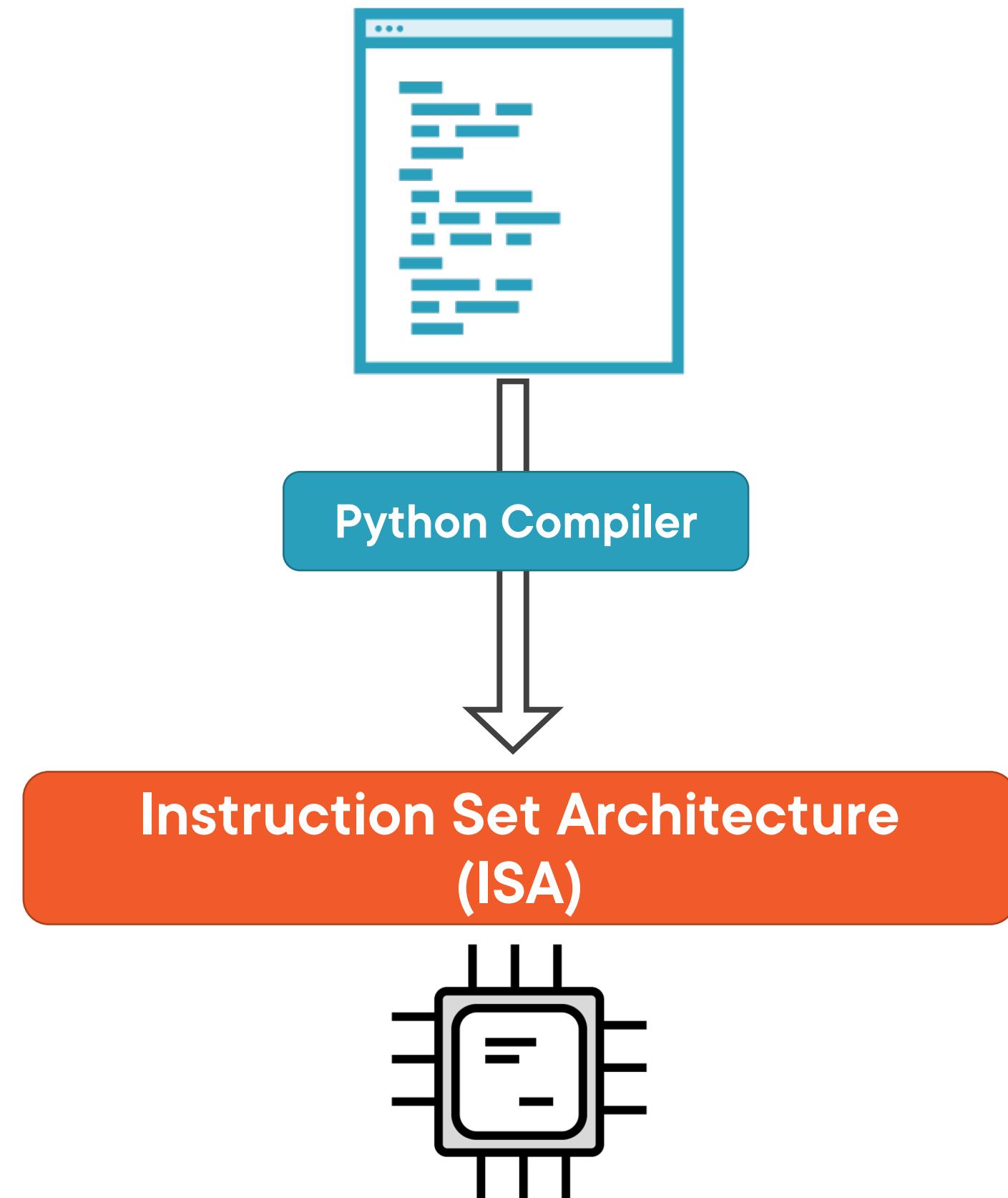


Python

Instruction Set Architecture  
(ISA)



# Python Can Be Compiled!

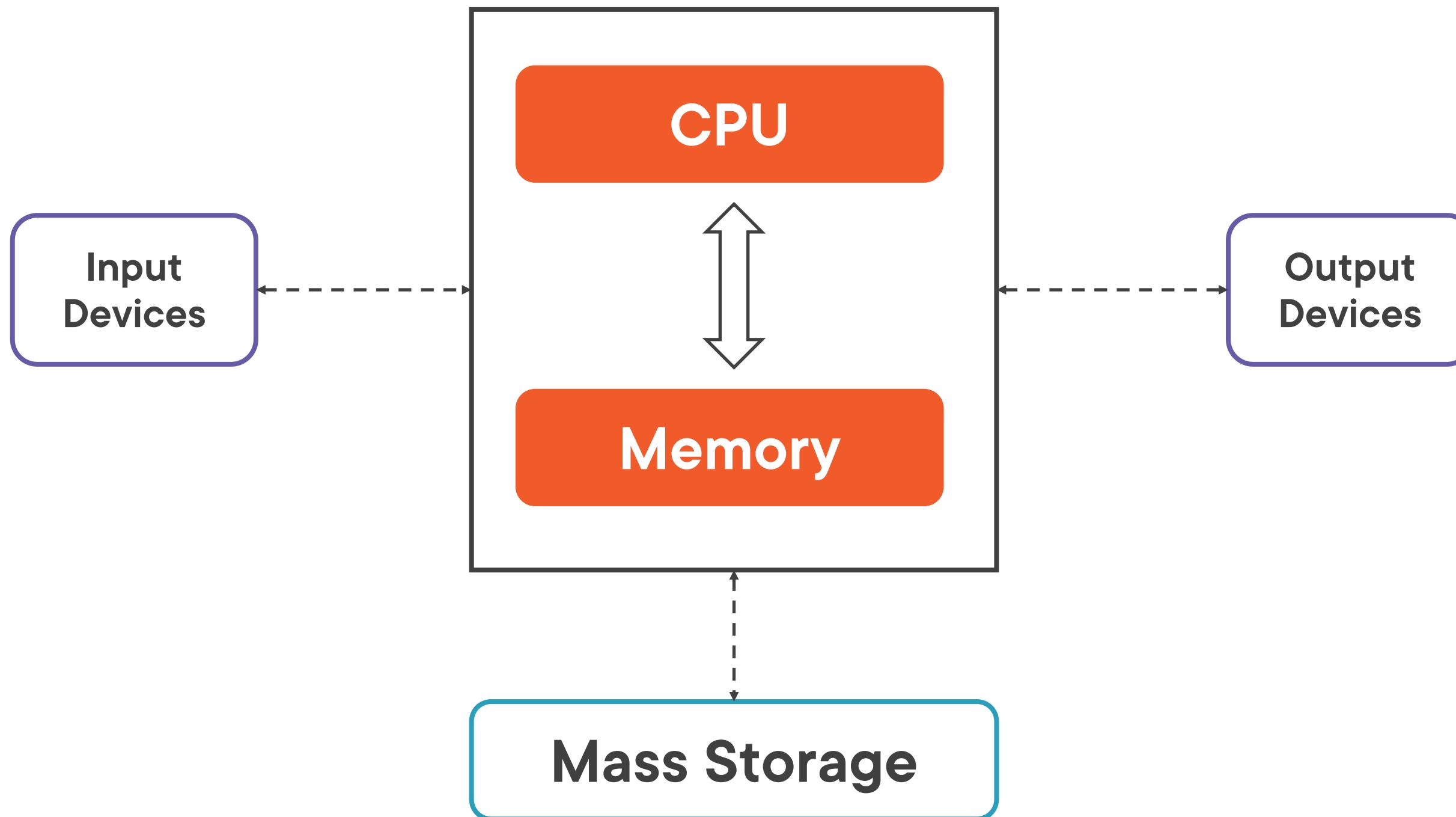


# Python Is Garbage-collected

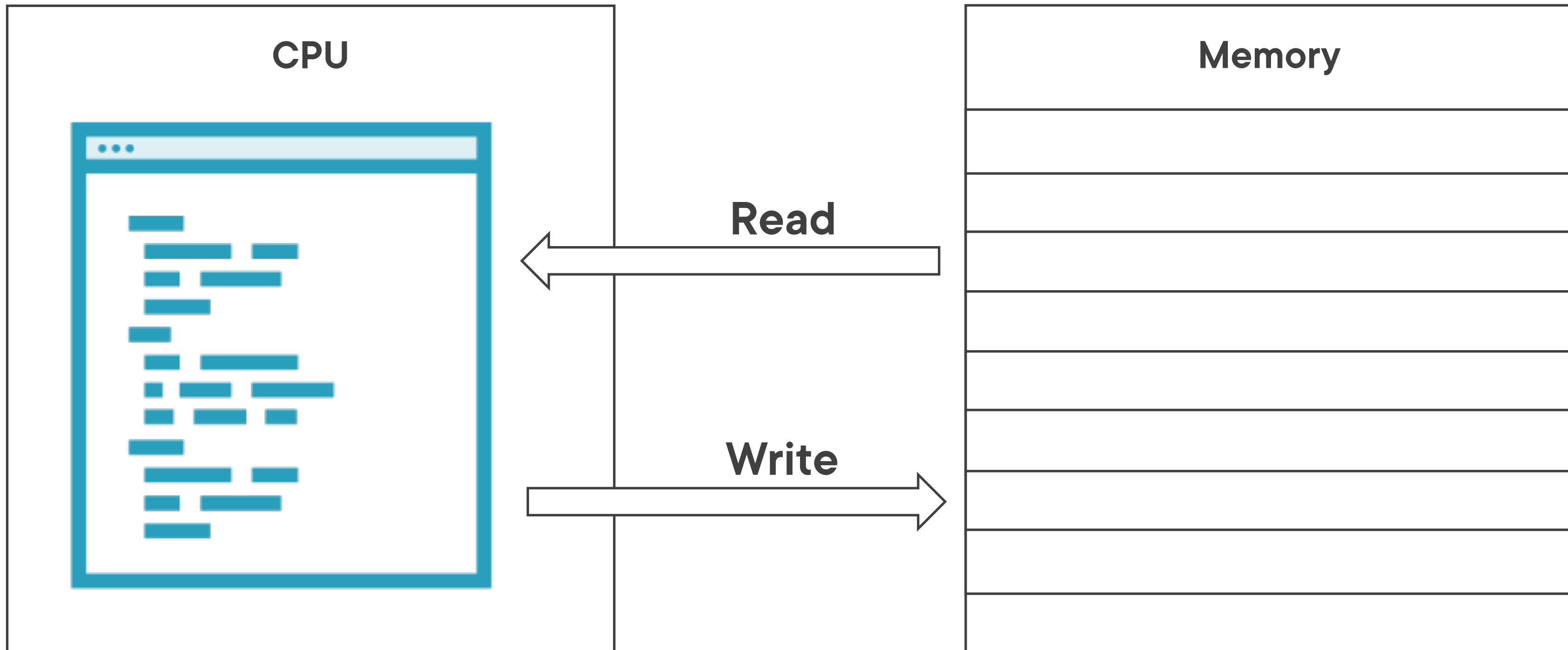
---



# von Neumann Architecture



# A Running Program





Clean it up  
yourself!

C/C++

Don't worry, I got this!

Python

# Why Care About Garbage Collection?



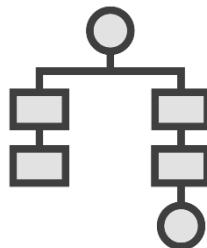
**No tedious memory bookkeeping**



**Avoid common memory leaks**



**Prevent whole classes of security issues**



**Efficiently implement certain persistent data structures**

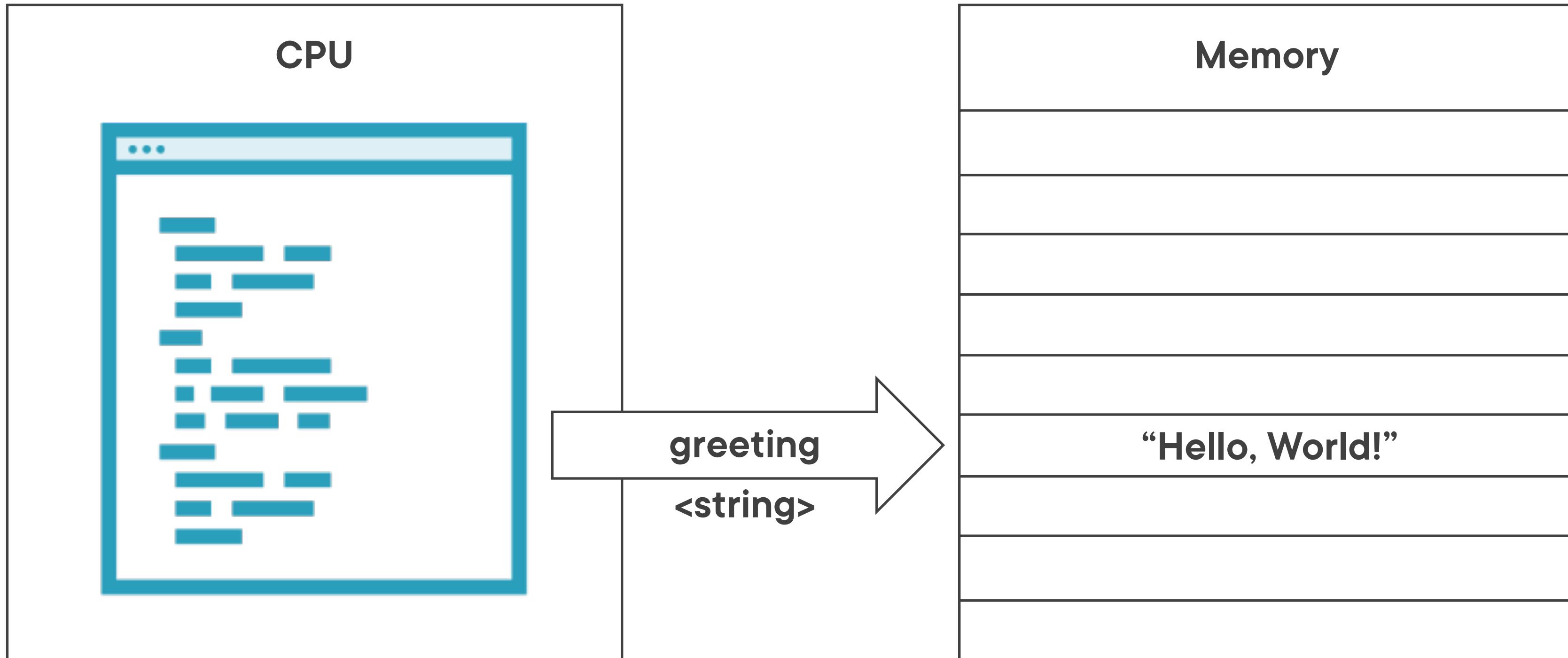


# Python Is Dynamically-typed

---



# A Running Program



# Static Typing

String  
Integer

...

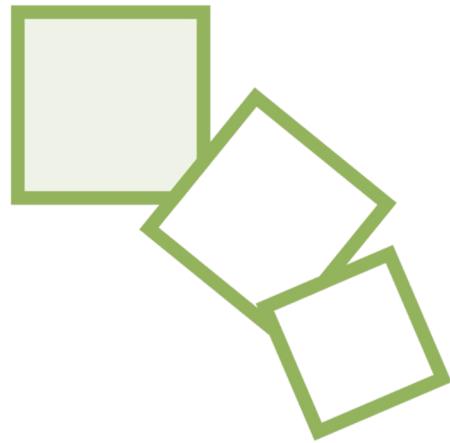
...

~~answer = "What is 42?"~~

```
greeting = "Hello, World!";  
answer = 42;
```



# Dynamic Typing



Type changing over lifetime of variable

```
answer = "What is 42?"  
...  
...  
answer = 42
```



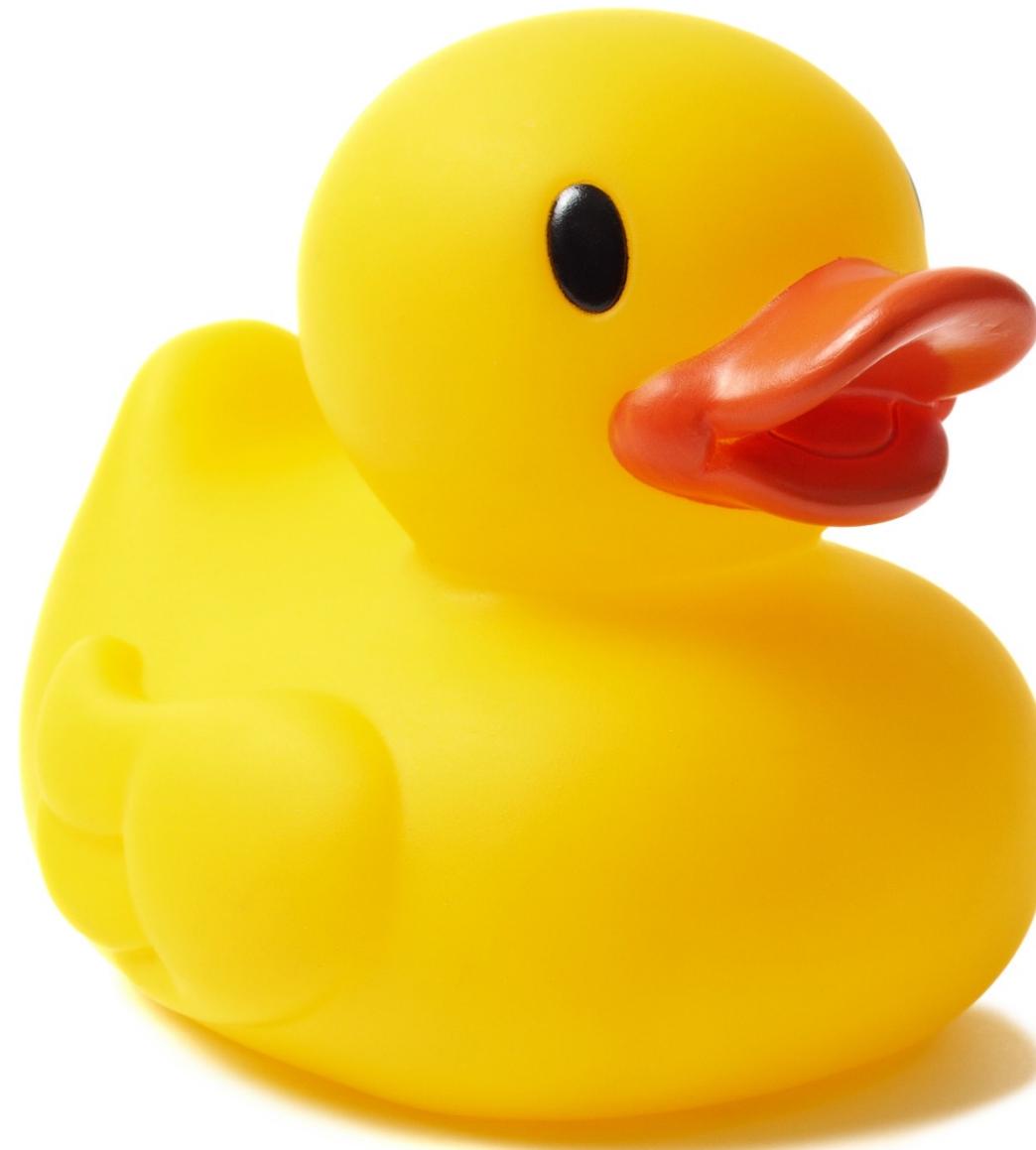
Type checking happens at runtime

```
>>> 42 + "Forty Two"
```

```
TypeError: unsupported operand  
type(s) for +: 'int' and 'str'
```



If it walks like a duck  
And it quacks like a duck  
It must be a duck!



```
class Duck:  
    def quack(self):  
        print "Quack, quack!"  
  
class Person:  
    def quack(self):  
        print "I'm a duck! Quack!"  
  
def feed_the_duck(thing):  
    thing.quack()  
  
feed_the_duck(Duck())  
feed_the_duck(Person())
```

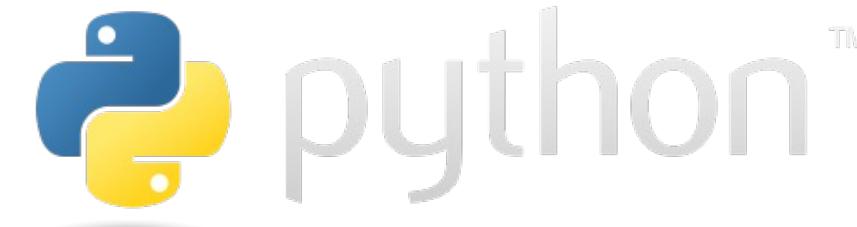


# So, What Is Python?

Dynamically-typed

Syntax

Garbage-collected



General-purpose

Interpreted

Multi-paradigm

