



keyword type built-in "string" number operator @decorator # comment

## HELLO, MOJO

```
def main():
    print("Hello, Mojo!")
```

Run it: `mojo hello.mojo`. Every program starts at `main()`.

## COMMENTS & DOCSTRINGS

```
# Line comment

def greet():
    """Docstring: what greet
    does."""
    print("hi")
```

## VAR OWNS, REF REFERS

```
var count = 0 # owned, inferred
    Int
var name: String = "Mojo"
count = count + 1 # var is
    mutable

var data = [1, 2, 3]
ref view = data[0] # ref to an
    element, no copy
view = 99 # writes through to
    data

comptime PI = 3.14159 # compile-
    time const
```

`var` declares an owned value, mutable by default. `ref` is a reference to a value it doesn't own. Mutable changes update the original value.

## BUILT-IN TYPES

```
Float32 == Scalar[DType.float32]
        == SIMD[DType.float32, 1]
```

<code>Int</code>	machine-word integer; default index type (not SIMD)
<code>UInt</code>	machine-word, SIMD-based
<code>Int8 ...</code>	sized integers
<code>Int64</code>	
<code>Float64</code>	default floating point (also Float32, Float16)
<code>Bool</code>	True / False
<code>String</code>	Unicode graphemes
<code>List[T]</code>	array, fixed size or growing
<code>SIMD[dt, n]</code>	n-wide numeric vector

Almost every numeric type is a **Scalar**: a length-1 **SIMD** vector. It looks like one number but it is really a vector, so scaling to vector math is built in. `Int` is the exception.

Types are **PascalCase** (`Int`); names are **lower\_snake\_case**.

## OPERATORS

<code>+ - * /</code>	add, subtract, multiply, divide
<code>// %</code>	floor divide, modulo
<code>**</code>	power (2 ** 10), also pow(2, 10)
<code>== !=</code>	equal, not equal
<code>&lt; &lt;= &gt; &gt;=</code>	comparisons (chainable)
<code>and or not</code>	logical, short-circuit
<code>+= -=</code>	compound assign ( <code>*=</code> , <code>/=</code> , ...)

`a < b < c` # chains to (a < b) and (b < c)

## STRINGS & PRINTING

```
var who = "Mojo"
print("Hi, " + who) #
    concatenation
print(t"Hi, {who}!") #
    interpolation
print(1, 2, 3, end=": ") #
    keyword args

var s = String(t"x = {1 + 1}") #
    to String
var raw = r"C:\path" # raw string
```

Mojo uses `t"..."` where Python uses `f"..."`. `print` takes a t-string directly; cast with `String(...)` to use one elsewhere. Triple quotes make multi-line strings.

## CONTROL FLOW

```
if x > 0:
    print("positive")
elif x == 0:
    print("zero")
else:
    print("negative")

# ternary
var kind = "even" if x % 2 == 0
    else "odd"
```

## LOOPS

```
for i in range(5): # 0 1 2 3 4
    print(i)

for item in [10, 20, 30]: #
    iterate an array
    if item == 20:
        continue # skip to next
    if item == 30:
        break # stop the loop
    print(item)

while n > 0: # loop while true
    print(n)
    n -= 1
```

Repeat n times with `for _ in range(n)` (`_` discards the value).

## FUNCTIONS

```
def add(a: Int, b: Int) -> Int:
  return a + b

def greet(name: String =
  "world"): # default
  print(t"Hi, {name}")

def risky() raises: # may raise
  raise Error("boom")

def nothing():
  pass # do-nothing body
```

No `->` means the function returns **None**.  
`def` can't raise unless marked **raises**.

## LISTS

```
var xs: List[Int] = [1, 2, 3]
xs.append(4)
print(xs[0]) # 1
print(len(xs)) # 4
```

## STRUCTS

```
@fieldwise_init # synthesizes
  __init__
struct Point:
  var x: Int
  var y: Int

struct Counter:
  var n: Int
  def __init__(out self): #
    builds self
    self.n = 0
  def bump(mut self): #
    modifies self
    self.n += 1

def main():
  var p = Point(3, 4)
  print(p.x, p.y) # 3 4
```

Every instance method takes **self** as its first argument. Structs also support **comptime** constants and static methods (`@staticmethod`).

## IMPORTS

```
from std.math import sqrt # one
  name
from std.math import sqrt as root
  # aliased
```

Built-ins like **Int**, **String**, **List**, and **print** are in the prelude, no import needed.

## COMING FROM PYTHON

- Every value has a fixed type. No implicit numeric conversion: write `Float64(n)`, `Int(x)`.
- Assignment copies the value; it is not a shared reference.
- String interpolation is `t"..."`, not `f"..."`.

## COMING FROM C++ / RUST

- Python-style syntax: indentation and `def`, no braces, semicolons, or headers.
- Value semantics with moves: `^` transfers ownership (like `std::move` or a Rust move); `__del__` gives RAII cleanup.
- Behavior comes from **traits**, not class inheritance or templates (like Rust traits or C++20 concepts).