

### @dataclass

```
from dataclasses import
dataclass
@dataclass
class my_class:
    field_one: type_one
    field_two: type_two
    field_three:
type_three
# automatically gets __init__,
__repr__, and __eq__
my_instance = my_class(
    field_one=value_
_one,
    field_two=value_
_two,
    field_three=va
ue_three
)
```

A decorator that automatically generates `__init__`, `__repr__`, and `__eq__` methods from class fields.

Each field is declared with its name and type annotation — no need to write a constructor manually.

Part of the standard library via `from dataclasses import dataclass`.

### io

```
import io
# in-memory binary stream
binary_buffer = io.BytesIO(b "some bytes")
binary_buffer.read() # read
all bytes
binary_buffer.read(4) # read
exactly 4 bytes from cursor
position
binary_buffer.write(b " -
more") # write bytes
binary_buffer.seek(0) # move
cursor back to start
# in-memory text stream
```

### io (cont)

```
> text_buffer = io.StringIO("some text")
text_buffer.read() # read all text
text_buffer.read(4) # read exactly 4
characters
text_buffer.seek(0) # move cursor back to
start
```

Python's standard library module for working with streams and file-like objects. Provides in-memory file objects that behave exactly like real files but exist only in RAM. BytesIO is for binary data, StringIO is for text data. `read(n)` reads exactly `n` bytes from the current cursor position — useful for parsing binary formats where you need to read fixed-size chunks.

### struct

```
import struct
# endianness prefix
# < = little-endian (least
significant byte first, most
common)
# > = big-endian (most signif -
icant byte first, used in
network protocols)
# type codes
# b = int8 B = uint8
# h = int16 H = uint16
# i = int32 I = uint32
# q = int64 Q = uint64
# f = float32 d = float64
# s = char[] ? = bool
# repeating type codes - t = any
type code
```

### struct (cont)

```
> # t = one value tt = two values ttt = three
values
# unpack — read bytes into Python values
value: int = struct.unpack("<i", f.read(4))[0] #
read one int32
a: int; b: int = struct.unpack("<ii", f.read(8)) #
read two int32s
three_ints: tuple = struct.unpack("<iii",
f.read(12)) # read three int32s
# pack — convert Python values into bytes
data: bytes = struct.pack("<i", 42) # write
one int32
data: bytes = struct.pack("<ii", 42, 100) #
write two int32s
```

Converts between Python values and C-style binary data using format strings. `struct.unpack(format, bytes)` reads bytes and returns a tuple of Python values. `struct.pack(format, values)` converts Python values into bytes.



By blakecromar

[cheatography.com/blakecromar/](http://cheatography.com/blakecromar/)

Not published yet.

Last updated 1st July, 2026.

Page 1 of 1.

Sponsored by [CrosswordCheats.com](http://CrosswordCheats.com)

Learn to solve cryptic crosswords!

<http://crosswordcheats.com>