Python (Standard Libraries) Cheat Sheet

by blakecromar via cheatography.com/218070/cs/48252/

@dataclass

```
from dataclasses import
dataclass
@dataclass
class my_class:
        fie ld_one: type_one
        fie ld_two: type_two
        fie ld_ three:
type_three
# automa tically gets __init__,
__repr__, and __eq__
my_ins tance = my_class(
        fie ld_ one =va lue -
_one,
        fie ld_ two =va lue -
_two,
        fie ld_ thr ee= val -
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 constr-uctor manually.

Part of the standard library via from datacl-asses import dataclass.

io

```
import io
# in-memory binary stream
binary _buffer = io.Byt esI -
O(b "some bytes")
binary _bu ffe r.r ead() # read
all bytes
binary _bu ffe r.r ead(4) # read
exactly 4 bytes from cursor
position
binary _bu ffe r.w rit e(b " -
mor e") # write bytes
binary _bu ffe r.s eek(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
signif icant 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.