PYTHON

CHEATSHEET

Every syntax explained — Types · Strings · Lists · Dicts · Classes · Async · Decorators · and more

188 entries · 15 categories

@contentintech

contentintech.com

▸ DATA TYPES

$ int, float, str, bool
Core scalar types

$ list, tuple, dict, set
Core collection types

$ frozenset, bytes, bytearray
Immutable set / byte sequences

$ NoneType (None)
Represents the absence of a value

$ type(x)
Return the type of object x

$ isinstance(x, int)
Check if x is an instance of int

$ int('42') float('3.14')
Type casting from string

$ str(42) bool(0) list((1,2))
Cast between types

$ complex(2, 3) # 2+3j
Complex number literal

$ x: int = 10 # type hint
Variable with type annotation

▸ STRINGS

$ f"Hello {name}!"
f-string interpolation (3.6+)

$ "{}".format(value)
str.format() interpolation

$ s.split(",") s.split()
Split on delimiter or whitespace

$ ",".join(["a","b"])
Join iterable with separator

$ s.strip() s.lstrip() s.rstrip()
Strip leading/trailing whitespace

$ s.replace("old", "new")
Replace all occurrences

$ s.find("sub") s.index("sub")
Find index; index raises if missing

$ s.startswith("hi") .endswith()
Prefix / suffix check → bool

$ s.upper() s.lower() s.title()
Case conversion methods

$ s.encode("utf-8") b.decode()
Encode str→bytes / decode bytes→str

$ """multi\nline"""
Triple-quoted multiline string

$ r"C:\\Users\\name"
Raw string — backslashes literal

$ s[1:5] s[::2] s[::-1]
Slice: start:stop:step

$ s.zfill(5) s.center(10, '-')
Pad / center string

$ s.isdigit() s.isalpha()
Character-class test methods

▸ LISTS

$ lst.append(x)
Add x to end of list

$ lst.extend([x, y])
Append all items from iterable

$ lst.insert(i, x)
Insert x at index i

$ lst.remove(x)
Remove first occurrence of x

$ lst.pop() lst.pop(i)
Remove & return last / index i

$ lst.sort() lst.sort(reverse=True)
Sort list in-place

$ sorted(lst, key=len)
Return new sorted list

$ lst.reverse() reversed(lst)
Reverse in-place / iterator

$ lst.index(x) lst.count(x)
Find index / count occurrences

$ lst.copy() lst[:]
Shallow copy of list

$ [x*2 for x in lst if x > 0]
List comprehension with filter

$ lst[1:4] lst[-3:] lst[::2]
Slice list

$ zip(a, b) list(zip(a,b))
Pair elements from iterables

$ enumerate(lst, start=1)
Index+value pairs

$ *a, last = lst # unpack
Extended unpacking

▸ DICTIONARIES

$ d.get(key, default)
Safe lookup with fallback

$ d.update({"k": v})
Merge another dict in-place

$ d.keys() d.values() d.items()
Views of keys / values / pairs

$ d.pop(key) d.pop(key, default)
Remove key and return value

$ d.setdefault(key, default)
Set if missing, always return val

$ {k: v for k,v in d.items()}
Dict comprehension

$ d1 | d2 d1 |= d2
Merge dicts (3.9+)

$ from collections import defaultdict
Dict with default factory

$ from collections import OrderedDict
Dict that remembers insertion order

$ d.clear() len(d)
Clear all items / count keys

$ key in d key not in d
Membership test

$ dict.fromkeys(["a","b"], 0)
Create dict from key list

▸ SETS

$ s.add(x)
Add element x to set

$ s.remove(x) # KeyError if missing
Remove x; raises if absent

$ s.discard(x) # safe
Remove x silently if missing

$ a | b a.union(b)
Union — all elements

$ a & b a.intersection(b)
Intersection — common elements

$ a - b a.difference(b)
Difference — in a, not in b

$ a ^ b a.symmetric_difference(b)
Elements in a or b, not both

$ a.issubset(b) a <= b
Check if a b

$ a.issuperset(b) a >= b
Check if a b

$ {x for x in lst if x > 0}
Set comprehension

$ frozenset([1, 2, 3])
Immutable set (hashable)

▸ CONTROL FLOW

$ if x > 0: ... elif x==0: ...
Conditional branching

$ else: ...
Default branch of if/elif

$ for item in iterable: ...
Iterate over iterable

$ for i in range(start, stop, step):
Iterate with numeric range

$ while condition: ...
Loop while condition is true

$ break # exit loop
Exit enclosing loop immediately

$ continue # next iteration
Skip to next iteration

$ pass # placeholder
No-op statement placeholder

$ match cmd: case "quit": ...
Structural pattern matching (3.10+)

$ val = x if condition else y
Ternary / conditional expression

$ if (n := len(lst)) > 10:
Walrus operator := (3.8+)

$ for i, v in enumerate(lst): ...
Loop with index and value

$ for a, b in zip(x, y): ...
Loop over paired iterables

▸ FUNCTIONS

$ def greet(name): return f"Hi {name}"
Define a function

$ def f(*args): ...
Accept any positional args as tuple

$ def f(**kwargs): ...
Accept any keyword args as dict

$ def f(a, b=10): ...
Parameter with default value

$ def f(*, key): ...
Keyword-only parameter

$ def f(pos, /, normal): ...
Positional-only parameter (/)

$ lambda x: x * 2
Anonymous one-expression function

$ def gen(): yield x
Generator function — yields values

$ yield from sub_gen()
Delegate to sub-generator

$ def f(x: int) -> str: ...
Function with type annotations

$ return a, b # returns tuple
Return multiple values as tuple

$ def f(a, b, /): ...
All positional-only params

▸ CLASSES

$ class Dog: ...
Define a class

$ def __init__(self, name): ...
Constructor / initializer

$ self.name = name
Instance attribute assignment

$ class Poodle(Dog): ...
Inheritance from Dog

$ super().__init__(name)
Call parent class constructor

$ @classmethod def create(cls):
Class method — receives cls

$ @staticmethod def helper():
Static method — no self/cls

$ @property def age(self): ...
Read-only computed property

$ def __str__(self): ...
Human-readable string repr

$ def __repr__(self): ...
Unambiguous developer repr

$ def __len__(self): ...
Support len() on instance

$ def __eq__(self, other): ...
Equality comparison operator

$ from dataclasses import dataclass
Auto-generate boilerplate

$ from abc import ABC, abstractmethod
Abstract base class support

▸ EXCEPTIONS

$ try: ... except ValueError: ...
Catch a specific exception

$ except (TypeError, KeyError): ...
Catch multiple exception types

$ except Exception as e: ...
Catch any exception, bind to e

$ else: ...
Runs if no exception was raised

$ finally: ...
Always runs (cleanup)

$ raise ValueError("bad value")
Raise an exception explicitly

$ raise RuntimeError from e
Chain exceptions (cause)

$ class MyErr(Exception): pass
Define a custom exception class

$ BaseException → Exception → ...
Built-in exception hierarchy

$ from contextlib import suppress
Silently ignore an exception

$ with suppress(FileNotFoundError):
Context manager exception ignore

▸ FILE I/O

$ open("f.txt", "r") # modes: r,w,a,b
Open file — default read text

```
$ with open("f") as fh: data=fh.read()
  Read entire file safely

$ fh.readlines()  fh.readline()
  Read all lines / one line

$ fh.write("text")  fh.writelines()
  Write string / list of strings

$ from pathlib import Path
  OO filesystem paths (3.4+)

$ Path("dir").mkdir(parents=True)
  Create directory tree

$ list(Path(".").glob("*.py"))
  Glob files by pattern

$ import shutil; shutil.copy(s, d)
  Copy file from src to dst

$ import json; json.load(fh)
  Deserialize JSON from file

$ json.dump(obj, fh, indent=2)
  Serialize object to JSON file

$ import csv; csv.reader(fh)
  Iterate CSV rows as lists

$ csv.DictReader(fh)
  CSV rows as dicts (header keys)

$ import pickle; pickle.dump(o, fh)
  Serialize Python object to file

$ os.path.exists(p)  os.path.join()
  Classic path utilities
```

▸ COMPREHENSIONS & GENERATORS

```
$ [x**2 for x in range(10)]
  List comprehension

$ {k: v for k, v in pairs}
  Dict comprehension

$ {x for x in lst}
  Set comprehension

$ (x**2 for x in range(10))
  Generator expression (lazy)

$ def gen(): yield x
  Generator function

$ next(gen_obj)
  Advance generator by one step

$ from itertools import chain
  Chain multiple iterables

$ from itertools import islice
  Lazy slice of an iterator

$ from itertools import product
  Cartesian product of iterables

$ from itertools import combinations
  r-length combinations

$ from itertools import permutations
  r-length permutations

$ from itertools import groupby
  Group consecutive elements
```

▸ MODULES & PACKAGES

```
$ import os  import sys
  Import standard library modules

$ from os import path, getcwd
  Import specific names

$ import numpy as np
  Import with alias

$ __all__ = ["func1", "Class1"]
  Declare public API of module

$ if __name__ == "__main__": ...
  Guard: only run as main script

$ pip install requests
  Install third-party package

$ python -m venv .venv
  Create a virtual environment

$ pip freeze > requirements.txt
  Export installed packages

$ pip install -r requirements.txt
  Install from requirements file

$ sys.path.append("/my/dir")
  Add directory to import search path

$ importlib.import_module(name)
  Dynamic import by string name
```

▸ CONCURRENCY

```
$ import threading
  OS-level threads module

$ t = threading.Thread(target=fn)
  Create a thread

$ t.start()  t.join()
  Start thread / wait for finish

$ import multiprocessing as mp
  True parallelism (bypasses GIL)

$ p = mp.Process(target=fn)
  Create a subprocess

$ import asyncio
  Event-loop based concurrency

$ asyncio.run(main())
  Run top-level async coroutine

$ async def fetch(): ...
  Define a coroutine function

$ result = await coro()
  Await a coroutine

$ asyncio.gather(c1, c2)
  Run coroutines concurrently

$ from concurrent.futures import ...
  ThreadPoolExecutor / Process

$ # GIL limits CPU threads; use mp
  GIL note for CPU-bound work
```

▸ BUILT-INS

```
$ len(x)  range(n)  enumerate(x)
  Length / range / indexed iter

$ zip(a,b)  map(fn,it)  filter(fn,it)
  Combine / transform iterables

$ sorted(x)  reversed(x)
  Sort / reverse (return iterator)

$ any(it)  all(it)
  True if any / all items truthy

$ sum(it)  min(it)  max(it)
  Aggregate numeric functions

$ abs(x)  round(x, n)  pow(x, y)
  Math operations

$ print(*args, sep, end, file)
  Print to stdout (or file)

$ input("prompt")  # returns str
  Read line from stdin

$ open(path, mode)  # see File I/O
  Open file object

$ type(x)  isinstance(x, T)
  Type inspection
```

```
$ dir(obj)  help(obj)
  List attributes / show docs
```

```
$ id(x)  hash(x)
  Object identity / hash value
```

```
$ repr(x)  str(x)
  Developer / user string repr
```

```
$ vars(obj)  getattr(obj, name)
  Attribute introspection
```

▸ DECORATORS & CONTEXT MANAGERS

```
$ def my_dec(fn):   @wraps(fn)
  Decorator preserving metadata
```

```
$ from functools import wraps
  Copy wrapped function metadata
```

```
$ @property  # read-only attr
  Property decorator on getter
```

```
$ @x.setter  # write property
  Setter for @property
```

```
$ from functools import cache
  Memoize with unbounded cache (3.9+)
```

```
$ from functools import lru_cache
  Memoize with max-size LRU cache
```

```
$ @lru_cache(maxsize=128)
  Apply LRU cache to function
```

```
$ from contextlib import contextmanager
  Turn generator into ctx mgr
```

```
$ @contextmanager def cm(): yield
  Yield-based context manager
```

```
$ class CM:   def __enter__(self):
  Class-based context manager
```

```
$   def __exit__(self, *exc): ...
  __exit__ handles exceptions
```

```
$ with open(f) as fh: ...
  Use context manager with with
```

STAY CONNECTED

Follow us for daily tech content, tutorials, and resources

INSTAGRAM

@contentintech

instagram.com/contentintech

YOUTUBE

@contenintech

youtube.com/@contenintech

WHATSAPP

Contentintech Channel

whatsapp.com/channel/0029VbCZJQU05MUijd2ofT1D

Thanks for learning with us — share this cheatsheet with your team!

contentintech.com