

JSON is a plain-text format for data (objects, lists, strings, numbers, true/false and null) that almost every language and web API can read. Python's json module turns JSON text into dicts and lists, and turns them back into text. Examples start with `import json`.

TEXT TO PYTHON AND BACK

The s in loads and dumps stands for string.

```
import json
```

```
json.loads('{ "name": "Ada", "age": 36 }')  
→ {'name': 'Ada', 'age': 36}  
loads: a JSON string becomes a Python dict
```

```
json.dumps({ "name": "Ada", "age": 36 })  
→ '{"name": "Ada", "age": 36}'  
dumps: Python data becomes a JSON string
```

```
json.dumps([1, "two", None, True])  
→ '[1, "two", null, true]'  
None becomes null, True becomes true
```

HOW THE TYPES MATCH

```
import json
```

dict	<->	object	{}
list	<->	array	[]
str	<->	string	""
int, float	<->	number	
True	<->	true	
None	<->	null	

each Python type and the JSON type it becomes

```
json.dumps((1, 2)) → '[1, 2]'  
a tuple becomes an array...
```

```
json.loads("[1, 2]") → [1, 2]  
...and the array comes back as a list
```

PRETTY AND SORTED

```
import json  
order = {"item": "tea", "id": 7}
```

```
print(json.dumps(order, indent=2))  
indent=2 puts each item on its own line, 2 spaces in per level
```

```
json.dumps(order, sort_keys=True)  
→ '{"id": 7, "item": "tea"}'  
puts the keys in A to Z order
```

```
json.dumps(order, separators=(",", ":"))  
→ '{"item": "tea", "id": 7}'  
the shortest text: no spaces
```

FILES: DUMP AND LOAD

```
import json  
orders = [{"name": "Grace", "total": 12.5}]
```

```
with open("orders.json", "w",  
encoding="utf-8") as f:  
    json.dump(orders, f, indent=2)  
dump (no s) writes the data to an open file
```

```
with open("orders.json", encoding="utf-8") as f:  
    data = json.load(f)
```

load (no s) reads a file into Python data

```
data == orders → True  
the same data came back
```

TEXT IN OTHER LANGUAGES

Both strings are valid JSON and load back as the same text. Write the file with encoding="utf-8".

```
import json
```

```
json.dumps("café") → '"caf\\u00e9"'  
by default, letters outside basic English become \u codes. Python shows the one backslash as two here.
```

```
json.dumps("café", ensure_ascii=False)  
→ '"café"'  
ensure_ascii=False keeps the real letters
```

DATES, SETS AND OTHER TYPES

```
import json  
from datetime import date
```

```
json.dumps({"day": date(2026, 10, 24)})  
→ raises TypeError  
json cannot write a date
```

```
json.dumps({"day": date(2026, 10, 24)},  
default=str)  
→ '{"day": "2026-10-24}"'  
json calls the default= function for any type it cannot write
```

```
json.dumps({"tags": {"vip"}},  
default=list)  
→ '{"tags": ["vip"]}'  
default=list saves a set as a list
```

```
json.loads('{ "day": "2026-10-24" }')  
["day"]  
→ '2026-10-24'  
the date comes back as a plain string...
```

```
date.fromisoformat("2026-10-24")  
→ datetime.date(2026, 10, 24)  
...so turn the string back into a date
```

MONEY AND EXACT NUMBERS

```
import json  
from decimal import Decimal
```

```
json.loads('{ "price": 19.99 }')['price']  
→ 19.99  
a number with a decimal point becomes a float
```

```
json.loads("19.99", parse_float=Decimal)  
→ Decimal('19.99')  
parse_float=Decimal keeps money values exact
```

JSON LINES: ONE RECORD PER LINE

```
import json  
rows = [{"user": "Hedy", "score": 91},  
{"user": "Linus", "score": 78}]
```

```
with open("log.jsonl", "w",  
encoding="utf-8") as f:  
    for row in rows:  
        f.write(json.dumps(row) + "\n")  
each line is one complete JSON value, so you can add lines and read them one at a time
```

```
with open("log.jsonl", encoding="utf-8")  
as f:  
    back = [json.loads(line) for line in f]  
read the file back, one line at a time
```

```
back  
→ [{'user': 'Hedy', 'score': 91},  
{'user': 'Linus', 'score': 78}]
```

FROM THE TERMINAL

```
python -m json.tool orders.json  
checks the file and prints it neatly. If the JSON is bad, it shows the line of the first error.
```

READ NESTED DATA SAFELY

```
import json  
data = json.loads('{ "user": { "name": "Ada",  
"langs": [ "py", "sql" ] } }')
```

```
data["user"]["langs"][0] → 'py'  
a key, then a key, then a list index
```

```
data["user"].get("email", "none")  
→ 'none'  
.get gives a default instead of a KeyError
```

```
data.get("order", {}).get("id") → None  
default {}: safe when a whole level is missing
```

BAD JSON

```
import json  
text = '{ "name": "Ada", }'
```

```
json.loads("") → raises JSONDecodeError  
an empty file or an empty web reply is not JSON
```

```
try:  
    data = json.loads(text)  
except json.JSONDecodeError as e:  
    print("Bad JSON, line", e.lineno)  
e.lineno and e.colno give the line and column of the error
```

WATCH OUT

```
import json
```

```
json.loads('{ "a": 1 }')  
→ raises JSONDecodeError  
single quotes are not valid JSON: use double quotes
```

```
json.loads(json.dumps({1: "a"}))  
→ {'1': 'a'}  
keys always become strings: 1 comes back as "1"
```

```
json.dump(data)  
TypeError: dump needs a file to write to. dumps gives a string.
```

```
json.loads('[1, 2, ]')  
→ raises JSONDecodeError  
JSON allows no comma after the last item
```

```
json.dumps(float("nan")) → 'NaN'  
NaN is not valid JSON, so other programs may reject it.  
allow_nan=False raises an error instead.
```