Bootcamp 134 | Python

Course 21 | Advanced Python

Amir Hossein Chegouniyan

Head of the Technical Team at Dariche Tejarat

Lecturer of Python – Django at Maktab Sharif

Amirhossein-chegounian

Content

Context Managers (with Statement)

- Principles of resource management using the with statement.
- Writing custom context managers with __enter__ and __exit__.
- Using built-in context managers for file handling and other common tasks.

Context Managers | Open

```
# File will be automatically closed after the block
with open("data.txt", "r") as f:
    content = f.read()
    print(content)
```

Context Managers | Simple Customize

```
class <MyContextName>:
    def __enter__(self):
        print("Entering context...")
        return "Some value to use inside block"

    def __exit__(self, exc_type, exc_value, traceback):
        print("Exiting context, cleaning up resources")

with <MyContextName> () as value:          # Use () after the class name
    print("Inside block:", value)
```

Python is the easier language to learn. No brackets, no main.
You get errors for writing an extra space

Context Managers | Decorator Customize

```
from contextlib import contextmanager


@contextmanager
def <my_context_name>():
    print("Entering context...")
    yield "Some value to use inside block"
    print("Exiting context, cleaning up resources")

with <my_context_name>() as value:
    print("Inside block:", value)
```

Iterators and Iterables

- Difference between iterables and iterators.
- Creating custom iterators using __iter__ and __next__.
- Exploring iter() and next() with Python’s built-in objects like lists, dictionaries, and ranges.

Iterators and Iterables | Difference

Feature	**Iterable**	**Iterator**
Definition	Object that can return an iterator	Object that returns items one by one
Main method(s)	__iter__()	__iter__() and __next__()
Usage	Can be looped over multiple times	Gets consumed once
Examples	list, tuple, str, dict, set, range	iter(list), file objects, generators

Iterators and Iterables | Using Iterator 1

```
nums = [1, 2, 3]
it = iter(nums)   # iterator from list


print(next(it))   # 1
print(next(it))   # 2
print(next(it))   # 3
# print(next(it)) # raises StopIteration
```

Iterators and Iterables | Using Iterator 2

```
my_dict = {"ali": 10, "ahmad": 20}
iterator = iter(my_dict)
while True:
    try:
        print(next(iterator))
    except:
        break
```

Typing in Python

- Introduction to type hints and annotations for better code clarity.
- Using advanced typing constructs like Union, List, Dict.
- Static type checking with tools like mypy.

Typing in Python | Introduction

```
def fib(n):
    a, b = 0, 1
    while a < n:
        yield a
        a, b = b, a+b
```

```
def fib(n: int) -> Iterator[int]:
    a, b = 0, 1
    while a < n:
        yield a
        a, b = b, a+b
```

Typing in Python | How to Use?

```
type Vector = list[float]


def scale(scalar: float, vector: Vector) -> Vector:
    return [scalar * num for num in vector]


# passes type checking; a list of floats qualifies as a Vector.
new_vector = scale(2.0, [1.0, -4.2, 5.4])
```

Typing in Python | See more …

- *Use this link for more study*

Variable Scopes

- Python's LEGB (Local, Enclosing, Global, Built-in) rule explained.
- Using global, local, and nonlocal keywords effectively.
- Practical examples of scoping with nested functions.

Variable Scopes | Python's LEGB

- Local (L):
  - Variables defined within the current function.
- Enclosing (E):
  - Variables in the scope of any outer functions (for nested functions).
- Global (G):
  - Variables defined at the top level of the module.
- Built-in (B):
  - Predefined names and functions that are built into the Python language.

Variable Scopes | nonlocal

- Local
- Global
- Nonlocal

APIs and Web Services

- Introduction to APIs: Understanding RESTful services, endpoints, and HTTP methods.
- Fetching and consuming APIs with Python's requests module.
- Error handling and response validation for API requests.

APIs and Web Services | Introduction

- Understanding RESTful services
- Endpoints
- HTTP methods.

APIs and Web Services | How to Work?

- Instal requests:
  - python –m pip install requests
- Use this link for practice
- Import requests to your code:
  - import requests
- Use from it in your code:
  - res = requests.get(url)
  - res.status_code
  - res.json()

JSON and XML Handling

- Parsing, reading, and writing JSON data using Python’s json module.
- Introduction to XML parsing using libraries like xml.etree.ElementTree.
- Comparison of JSON and XML for data representation.

JSON and XML Handling | Differences

Feature	**JSON**	**XML**
Readability	Simpler and shorter	Longer and tag-heavy
Structure	Key–value pairs, arrays	Tags and attributes
Main usage today	Data exchange in APIs (REST, GraphQL)	Older systems, configurations, documents (SOAP, RSS)
Data support	Strings, numbers, booleans, null, lists, objects	Everything as text (needs parsing for data types)
Size	Lighter (fewer extra characters)	Heavier
Schema	Simpler (JSON Schema)	More powerful (XSD, DTD)

JSON and XML Handling | How to use it?

- import json
- new_dict = json.loads(<a_json>)
- new_json = json.dumps(<a_dict>)

Any question?

Next course

- Introduction to Databases
- SQL Basics
- Filtering Data
- Customizing Results
- Grouping and Aggregation
- SQL Tools and Interfaces