Context Manager. Serialization

IT Academy

softserve

Agenda

- Context Manager
- @contextmanager
- Data serialization
- Object serialize / deserialize
- Json format
- CSV format

softserve

Context Manager

Context manager an object designed to be used in a with-statement.

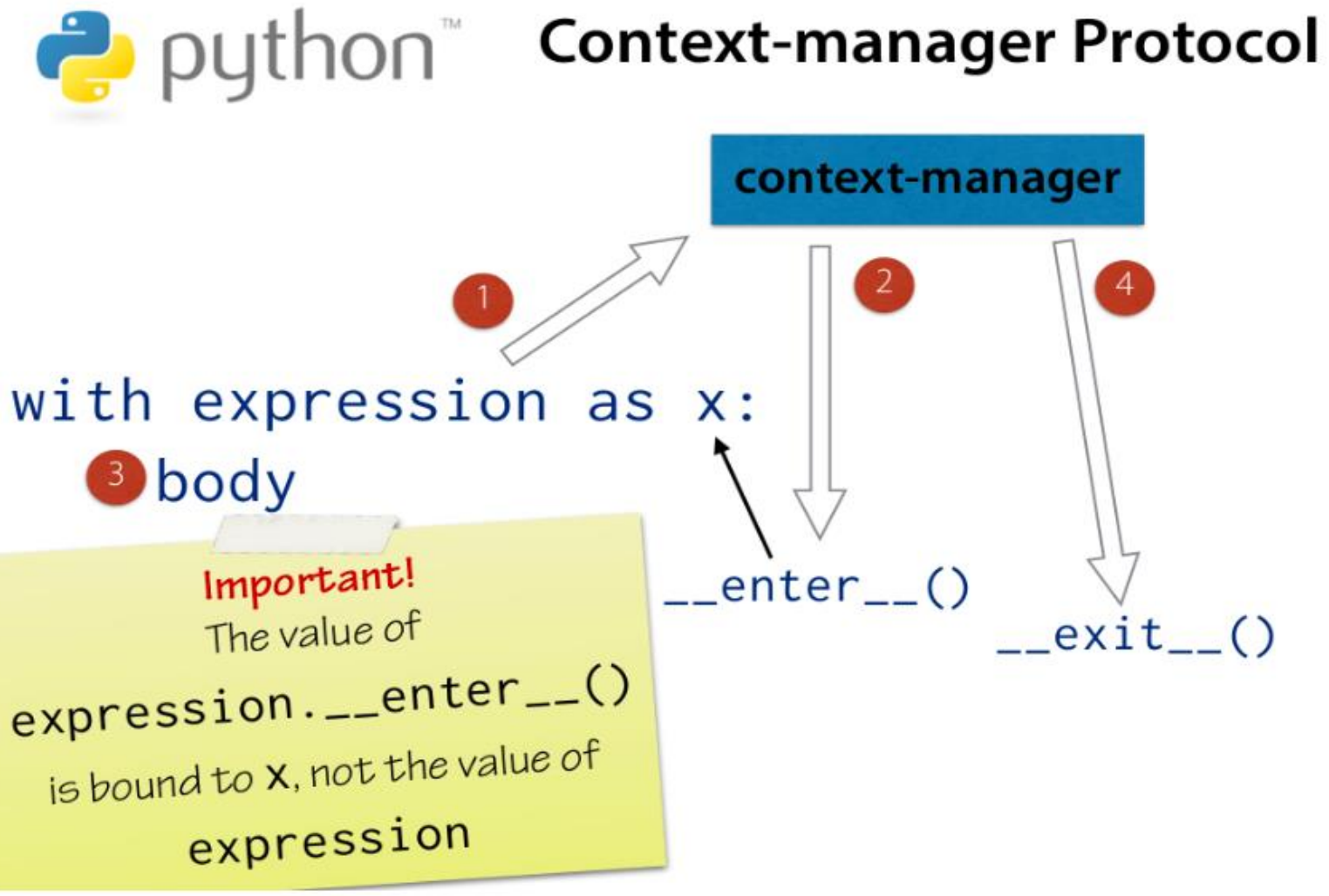
with *expression* [**as** *target*] ("," *expression* [**as** *target*])* **:**
    suite

```
with A() as a, B() as b:
    suite
```

```
with A() as a:
    with B() as b:
        suite
```

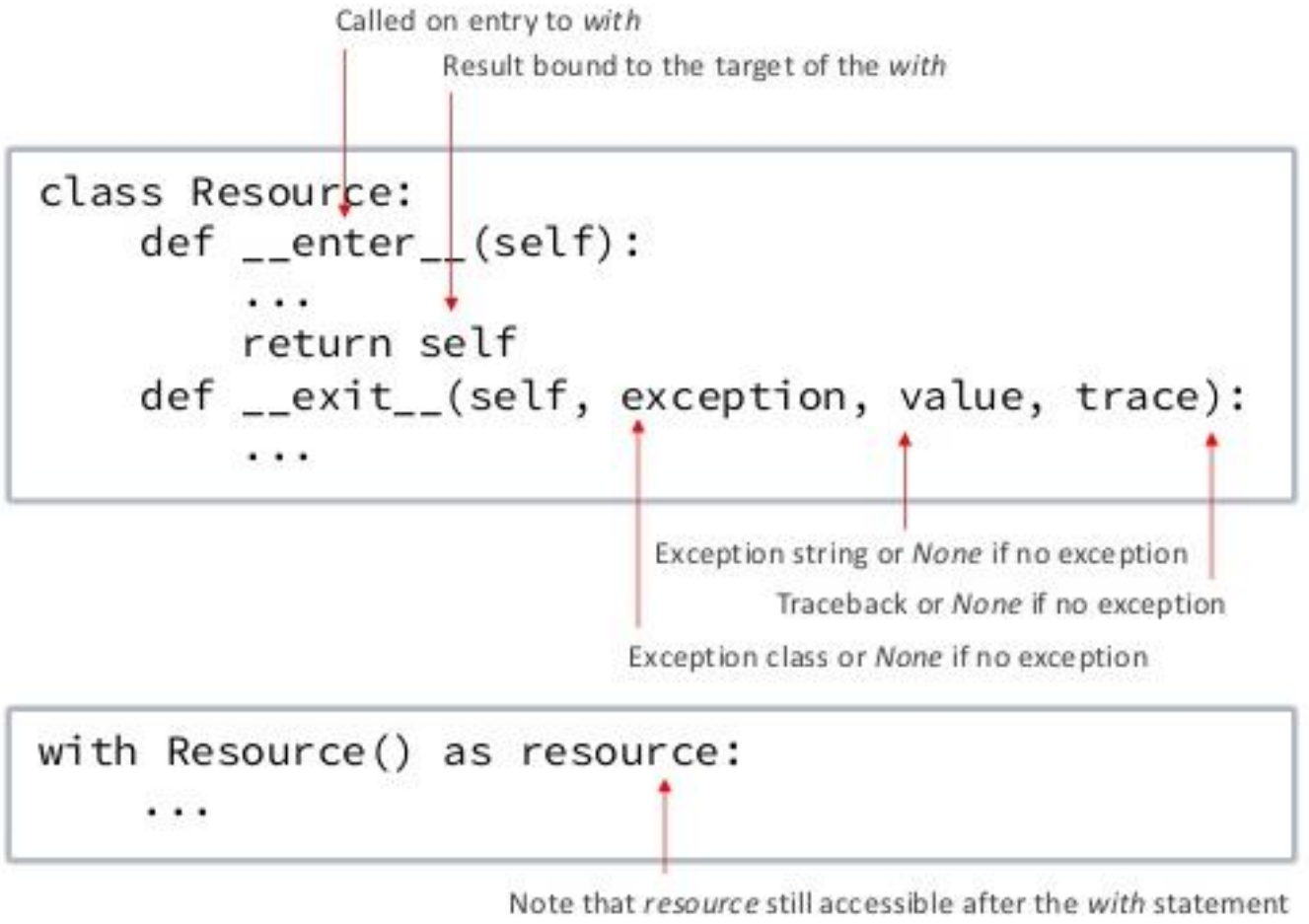
softserve

Context Manager



softserve

Context Manager Anatomy



softserve

Custom context manager

```
context manager 1 entered
in with block for context manager 1
context manager 1 exited
context manager 2 entered
context manager 2 exited
but with Exception with name My error
Error occurred.
```

```
class MyContextManager:
    def __init__(self, name):
        self.name = name

    def __enter__(self):
        print(f'{self.name} entered')
        return self.name

    def __exit__(self, *args):
        print(f'{self.name} exited')
        if args[0]: print(f'but with {args[0].__name__}
with name {args[1]}')
```

```
pcm1 = MyContextManager('context manager 1')
pcm2 = MyContextManager('context manager 2')

with pcm1 as name:
    print(f'in with block for {name}')

try:
    with pcm2 as name:
        raise Exception("My error")
        print(f'in with block for {name}')
except:
    print(f'Error occurred.')
```

softserve

With statement for file

Before:

```
file = open('welcome.txt', 'w')
try:
    file.write('hello world')
finally:
    file.close()
```

After:

```
with open("welcome.txt") as file:
  data = file.read()
```

```
with open('output.txt', 'w') as file:
  file.write('Hi there!')
```

softserve

Contextlib Utility contextmanager

```
@contextmanager
def some_generator(<arguments>):
    <setup>
    try:
        yield <value>
    finally:
        <cleanup>
```

```
with some_generator(<arguments>) as
<variable>:
    <body>
```

```
    <setup>
    try:
        <variable> = <value>
        <body>
    finally:
        <cleanup>
```

softserve

Manage file work with @contextmanager

```
from contextlib import contextmanager

@contextmanager
def managed_file(name, method):
 f = open(name, method)
 try:
    yield f
 finally:
    f.close()

with managed_file('hello.txt', "w") as f:
  f.write('hello, world!')
  f.write('bye now')
```

softserve

```
@contextmanager
def managed_block(name):
    f = name
    print(f'{name} entered')
    try:
        yield f
    except Exception as ex:
        print(f'{name} exited')
        print(f'but with
{ex.__class__.__name__} with name {ex}')
        raise Exception(ex)
    else:
        print(f'{name} exited')
```

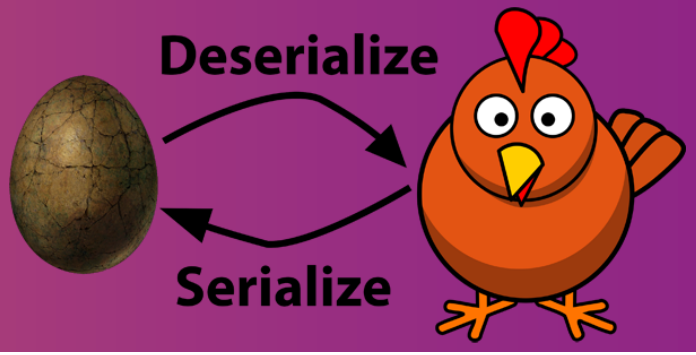
```
pcm1 = managed_block('context manager 1')
pcm2 = managed_block('context manager 2')

with pcm11 as name:
    print(f'in with block for {name}')

try:
    with pcm2 as name:
        raise Exception("My error")
        print(f'in with block for {name}')
except:
    print(f'Error occurred.')
```

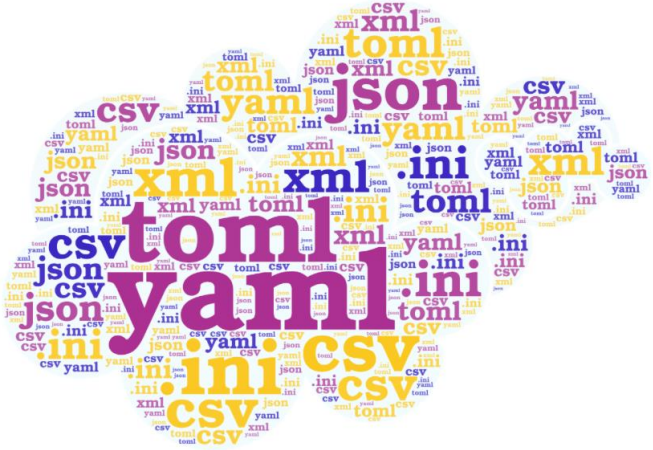
softserve

Data serialization



This process of turning an arbitrary Python object into a series of bytes is called **serializing** the object.

The byte stream representing the object can then be transmitted or stored, and later reconstructed to create a new object with the same characteristics.



softserve

Standard modules

dumps, dump, load, loads

VS

```
with open('/tmp/file.json', 'rb') as f:
  data = json.load(f)

with open('/tmp/file.json', 'wb') as f:
  json.dump(data, f)
```

for different languages

Pickle:

```
import pickle
grades = { 'Alice': 89, 'Bob': 72, 'Charles': 87
}

serial_grades = pickle.dumps( grades )

received_grades = pickle.loads(
serial_grades )
```

The pickle module is not intended to be secure against erroneous or maliciously constructed data. Never unpickle data received from an untrusted or unauthenticated source.

softserve

Serialize / deserialize of object

Data is in name/value pairs
Data is separated by commas

Curly braces hold objects
Square brackets hold arrays

Python	JSON
dict	object
list, tuple	array
str	string
int, float, int- & float-derived Enums	number
True	true
False	false
None	null

JSON	Python
object	dict
array	list
string	str
number (int)	int
number (real)	float
true	True
false	False
null	None

softserve

Object serialize / deserialize

```
class Student(object):
    def __init__(self, first_name: str, last_name: str):
        self.first_name = first_name
        self.last_name = last_name
    @classmethod
    def from_json(cls, data):
        return cls(**data)


class Team(object):
    def __init__(self, students: []):
        self.students = students
    @classmethod
    def from_json(cls, data):
        students = list(map(Student.from_json, data["students"]))
        return cls(students)
student1 = Student(first_name="Jake", last_name="Foo")
student2 = Student(first_name="Jason", last_name="Bar")
team = Team(students=[student1, student2])
# Serializing
data = json.dumps(team, default=lambda o: o.__dict__, sort_keys=True,
indent=4)
print(data)
# Deserializing
decoded_team = Team.from_json(json.loads(data))
print(decoded_team)
print(decoded_team.students)
```

softserve

JSON schema validation

import jsonschema

```
schema = {
    "type" : "object",
    "properties" : {
        "name" : {"type" : "string"},
        "age" : {"type" : "number"}
    },
    "required": ["age","name"]
}
```

```
  "type": "array",
  "items": {
    "type": "number"  }
}

validJson = {"name" : "Eggs", "age" : 10}
invalidJson1 =  {"name" : "Eggs", "age":”10"}
invalidJson2 =  {"name" : "Eggs"}
validate(validJson, schema)
validate(invalidJson1, schema)
validate(invalidJson2, schema)
```

None
’10’ is not for type ‘number
‘age’ is a required property

softserve

Read/Write .csv files

Main rules:

Each record is one line ...but: fields may contain embedded line-breaks so a record *may* span *more* than one line.

Fields are separated with commas.
Leading and trailing space-characters adjacent to comma field separators are ignored.
Fields with embedded commas must be delimited with double-quote characters.
Fields that contain double quote characters must be surounded by double-quotes, and the embedded double-quotes must each be represented by a pair of consecutive double quotes.
A field that contains embedded line-breaks must be surounded by double-quotes

Python CSV Module

```
import csv
with open('X:\data.csv','rt')as f:
    data = csv.reader(f)
    for row in data:
        print(row)
```

```
['hostname', 'vendor', 'model', 'location']
['sw1', 'Cisco', '3750', 'London']
['sw2', 'Cisco', '3850', 'Liverpool']
['sw3', 'Cisco', '3650', 'Liverpool']
['sw4', 'Cisco', '3650', 'London']
```

```
import csv
with open('X:\data.csv', mode='w') as file:
    writer = csv.writer(file, delimiter= ' , ')
    writer.writerow(['Name', 'birthday month'])
    writer.writerow(['Guido van Rossum', 'Jan'])
    writer.writerow(['James Gosling', 'Feb'])
```

DictWriter

writeheader()
writerows()

```
mydict =[{'name': 'Nikhil', 'year': '2'},
        {'name': 'Sanchit', 'year': '2'},
        {'name': 'Aditya', 'year': '2'},
        {'name': 'Sagar', 'year': '1'},
        {'name': 'Prateek', 'year': '3'},
        {'name': 'Sahil', 'year': '2'}]
```

```
fields = ['name', 'year']
filename = "university_records.csv"

with open(filename, 'w') as csvfile:
    writer = csv.DictWriter(csvfile, fieldnames = fields)
    writer.writeheader()
    writer.writerows(mydict)
```

.xml format

Main rules:

XML is a markup language which is designed to store data.

It is case sensitive.

XML offers you to define markup elements and generate customized markup language.

The basic unit in the XML is known as an element.

The XML language has no predefined tags.

It simplifies data sharing, data transport, platform changes, data availability Extension of an XML file is .xml

Libraries for xml parsing

```
<data>
 <items>
<item name="item1">item1abc</item>
<item name="item2">item2abc</item>
</items>
 </data>
```

```
from xml.dom import minidom
mydoc = minidom.parse('items.xml')
items =
mydoc.getElementsByTagName('item')
```

```
print('Item #2 attribute:')
print(items[1].attributes['name'].value)
print('\nAll attributes:')
for elem in items:
    print(elem.attributes['name'].value)
print('\nItem #2 data:')
print(items[1].firstChild.data)
print(items[1].childNodes[0].data)
print('\nAll item data:')
for elem in items:
    print(elem.firstChild.data)
```

softserve

```
pip install xmltodict
import xmltodict
import json
xml='''<root>
<name>First Name</name>
<article>Article 1</article>
<message>Message1</message>
<message>Message2</message>
</root>'''
my_dict=xmltodict.parse(xml)
json_data=json.dumps(my_dict)
print(json_data)
```

softserve

Thanks for attention!

softserve


TO THE FUTURE
softserve