# API and Python training

Session 7

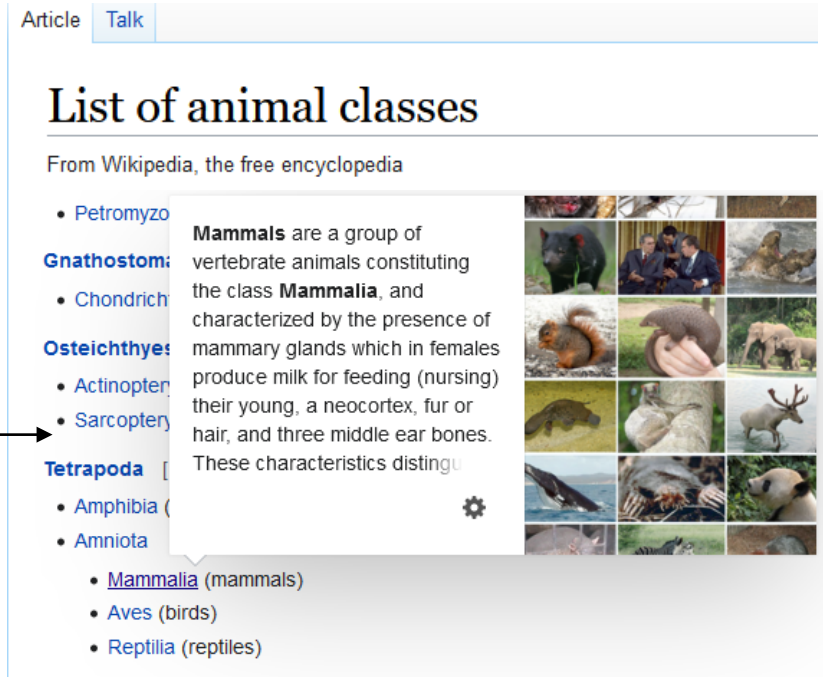
# This session agenda

- Recap from last session
- Classes and objects in real life
- Classes and objects in Python
- Constructors
- Class methods
- Class variables
- Object-oriented programming concepts
- Object-oriented programming example
- Demo

# Recap from Session 6 (Demo)

# Classes and objects in real life

- There are many 'things' around us and people use classification to group these things
- These things are logically grouped together using some **common properties**
- For example, animals are grouped into classes based on a few common traits.
- If you come across an animal you don't know yet, but you know it's a mammal, you can already assume it can produce milk and has fur
- A **Class** itself is a definition – describes what properties its instances have
- A **Class Instance** or **Object** is a real 'thing' which belongs to a particular class. We set some values to the properties thus fully describing the object
- **Your cat** is an object with some values assigned to its properties – property colour, value = black; name, number of legs (with default value of four for mammals)
- **My cat** is a different object with different properties, so different colour, name, potentially different number of legs (but not more than four)
- These are two different objects of the same class
- Not only these objects have some common traits, but they do the **same actions** – they can run, but can't fly
- We define in classes not only properties, but also possible actions




https://en.wikipedia.org/wiki/List_of_animal_classes

Two instances of class mammals with different properties:

# Classes and objects in Python

- To define a class use syntax:

  class <class_name>:

  <block of code>


https://en.wikipedia.org/wiki/Camel_case

- Naming convention for class names is **upper camel case**:

  *ResponseReceived , FirewallZones, etc*

- The block of code within the class has **indentation** to define where the class starts and ends
- Similar to variables in functions, you can/define variables within the class – these are called **class attributes (or class properties)**
- To **create a new object (also called class instance)** of a particular class, use syntax

  <variable> = <ClassName>()

- Then you work with different objects using their variable names/IDs
- To set or get object attribute use <obj_variable>.<attribute>

```python
# Define class
class Cat():
    name = ''
    fur_colour = ''

# create objects
my_cat = Cat()
your_cat = Cat()

# assign values to attributes
my_cat.name = 'Nika'
your_cat.name = 'Zelda'

my_cat.fur_colour = 'black'
your_cat.fur_colour = 'grey'

print(f'We have two object of class Cat: '
      f'{my_cat.name} is {my_cat.fur_colour} and '
      f'{your_cat.name} is {your_cat.fur_colour}')
```

classes00

```
We have two object of class Cat: Nika is black and Zelda is grey
```

# Classes and objects - definitions

- We group anything into classes, define properties (**class attributes or class properties**) and actions which a 'thing' can do or can be done on it (**class methods**)
- Class attributes are **variables**, class methods are **functions**
- Example of classes – definition of 'things':

Class router:

Attributes – *hostname, mgmt. IP, list of interfaces, list of routes, etc.*
Methods – *shutdown interface,*
*assign IP to interface (change attribute),*
*send routing update (request to change other object's attribute)*

Class host:

Attributes – *hostname, amount of RAM, CPU, disk, HDD, OS*
Methods – *attach disk, install new OS*

- **Objects = class instances**
- Define objects - real routers:
  Object id - *host1* Attributes – *dc01prod01, 8, 4, 200, linux*
  Object id – *host2* Attributes – *dc01prod02, 16, 16, 100, windows*
- Then we can change properties of these objects separately

and independently from any other objects

classes_and_objects.py

```python
class Host:
    name = ''
    list_of_interfaces = []
    cpu_number = 0
    installed_os = ''


dc1host = Host()
dc2host = Host()

dc1host.name = 'dc01prodhost01'
dc2host.name = 'dc02prodhost02'

print(f'Host names: {dc1host.name} and {dc2host.name}')
```

Host

classes_and_objects

```
Host names: dc01prodhost01 and dc02prodhost02
```

# Constructors

- Often, instead of creating a new object and assigning some values later, it's convenient to define some attributes **at the same time** we create an object
- To set these initial values (or do some initial actions) use special **function** called **constructor** – this function will be executed automatically **every time you create a new object**
- Syntax:
  *class <name>:*
  *def __init__(self, <parameters>):*
  *<block of code>*

- __init__ is a reserved name of function-constructor
- Note the first argument of any class function is always ***self***

https://www.geeksforgeeks.org/self-in-python-class/

- The rest of arguments are normal arguments similar in all other Python functions
- To access class variables within the constructor, use *self.<variable_name>* *See Lines 9-12* ---------->
- Once you defined a constructor, you can pass parameters to the constructor function and set object's attributes using these parameters - *Lines 15 and 16* -------->

```
1  class Host:
2      name = ''
3      list_of_interfaces = []
4      cpu_number = 0
5      installed_os = ''
6
7      # constructor
8      def __init__(self, name, cpu_number, list_of_interfaces=[], installed_os='linux'):
9          self.name = name
10         self.cpu_number = cpu_number
11         self.list_of_interfaces = list_of_interfaces
12         self.installed_os = installed_os
13
14
15 dc1host = Host('dc01prodhost01', 4, ['eth0', 'eth1'], 'ubuntu20.04')
16 dc2host = Host('dc02prodhost02', 8, [], 'windows10')
17
18 print(f'Host {dc1host.name} has {dc1host.cpu_number} CPU and OS {dc1host.installed_os}')
19 print(f'Host {dc2host.name} has {dc2host.cpu_number} CPU and OS {dc2host.installed_os}')
```

Host

classes_and_objects

```
Host dc01prodhost01 has 4 CPU and OS ubuntu20.04
Host dc02prodhost02 has 8 CPU and OS windows10
```

# Class methods

- **Class methods** are functions defined within the class, constructor is one of them
- The same syntax as all other functions
- The first argument is always **self**
- The class method can access class variables using syntax *self.<attribute>*

- Once we have created an object, we can use its methods, so class methods become object's methods (what this object can do)

- To call object's methods, syntax is the same as we access class variables, so <object>.<method>

- Note in the example - Even though we use the same method (attach_interface), we work with different objects, so we change attributes of each object independently

```python
class Host:
    # class variables
    name = ''
    list_of_interfaces = []
    cpu_number = 0
    installed_os = ''

    # constructor
    def __init__(self, name, cpu_number, list_of_interfaces=[], installed_os='linux'):
        self.name = name
        self.cpu_number = cpu_number
        self.list_of_interfaces = list_of_interfaces
        self.installed_os = installed_os

    # class methods
    def attach_interface(self, interface_name):
        self.list_of_interfaces.append(interface_name)


dc1host = Host('dc01prodhost01', 4, ['eth0', 'eth1'], 'ubuntu20.04')
dc2host = Host('dc02prodhost02', 8, [], 'windows10')

print(f'Host {dc1host.name} has {dc1host.cpu_number} CPU and interfaces {dc1host.list_of_interfaces}')
print(f'Host {dc2host.name} has {dc2host.cpu_number} CPU and interfaces {dc2host.list_of_interfaces}')

print('Attaching new interfaces...')
dc1host.attach_interface('eth2')
dc2host.attach_interface('eth0')

print(f'Host {dc1host.name} has {dc1host.cpu_number} CPU and interfaces {dc1host.list_of_interfaces}')
print(f'Host {dc2host.name} has {dc2host.cpu_number} CPU and interfaces {dc2host.list_of_interfaces}')
```

classes_and_objects

```
Host dc01prodhost01 has 4 CPU and interfaces ['eth0', 'eth1']
Host dc02prodhost02 has 8 CPU and interfaces []
Attaching new interfaces...
Host dc01prodhost01 has 4 CPU and interfaces ['eth0', 'eth1', 'eth2']
Host dc02prodhost02 has 8 CPU and interfaces ['eth0']
```

# Class variables

- Not necessary to define all variables on the class level

```python
class Host:
    name = ''
    list_of_interfaces = []
    cpu_number = 0
    installed_os = ''

    # constructor
    def __init__(self, name, cpu_number, list_of_interfaces=[], installed_os='linux'):
        self.name = name
        self.cpu_number = cpu_number
        self.list_of_interfaces = list_of_interfaces
        self.installed_os = installed_os
```

- The common practice is to define and initialise them in the constructor

- In the example below we introduce a new class variable – *mgmt_interface* type dictionary, but there might be a better way using object-oriented approach (next slides)

```python
class Host:
    # constructor
    def __init__(self, name, cpu_number, mgmt_intf):
        self.name = name
        self.cpu_number = cpu_number
        self.mgmt_interface = mgmt_intf

# define mgmt interface
mgmt_intf_host1 = {'name': 'mgmt0', 'ip': '10.10.10.1'}

# create object
dc1host = Host('dc01prodhost01', 4, mgmt_intf_host1)

print(f'Host {dc1host.name} has '
      f'mgmt interface {dc1host.mgmt_interface["name"]} '
      f'with ip address {dc1host.mgmt_interface["ip"]}')
```

# Object-oriented programming

- Treat ‘things’ you work with, as class instances: define a class with attributes and methods, create an object and work with the objects
- All variables (and even functions) in Python are objects, so we can use their attributes and methods
- This approach is called Object-oriented programming (OOP)
- IDE can list available methods and attributes for an object:

```
my_string = 'test'
my_string.
    print                                   print(expr)
    lower(self)                                     str
    __doc__                                         str
    format(self, args, kwargs)                      str
    __module__                                      str
    capitalize(self)                                str
    casefold(self)                                  str
    center(self, __width, __fillchar)               str
    count(self, x, __start, __end)                  str
    encode(self, encoding, errors)                  str
    endswith(self, suffix, start, end)              str
```

```
response = requests.post(base_url + 'system/api/v1/auth/token')
if response.status_code == 200:
    print(response.text)
    print(response.)
        text                                    Response
        status_code                             Response
        json(self, kwargs)                      Response
        reason                                  Response
        content                                 Response
        apparent_encoding                       Response
        close(self)                             Response
        cookies                                 Response
        elapsed                                 Response
        encoding                                Response
        headers                                 Response
        history                                 Response
Press Ctrl+. to choose the selected (or first) suggestion and insert a dot afterwards  Next Tip
```

https://en.wikipedia.org/wiki/Object-oriented_programming

# Object-oriented programming - example

- Represent 'things' as classes with attributes
- A class can have a variable of another class
- To access these nested class variables or methods, use <object>.<object>.attribute<etc>
- Example

Line 9 – *mgmt_interface* of class *Router* is a class instance of *Interface*

Line 13 – we can get Interface name using the following construct:

```
router1.mgmt_interface.name.upper()
^^^     ^^^           ^^^   ^^^^
```

| Variable of class Router | Variable of class Interface | Variable of class string | Method – note () of class string |
|---|---|---|---|

```
1  class Interface:
2      def __init__(self, name, ip):
3          self.name = name
4          self.ip_addr = ip
5
6  class Router:
7      def __init__(self, name, mgmt_interface_name, mgmt_interface_ip):
8          self.name = name
9          self.mgmt_interface = Interface(mgmt_interface_name, mgmt_interface_ip)
10
11 router1 = Router('prodrtr1', 'eth0', '10.1.2.2')
12
13 print(router1.mgmt_interface.name.upper(), router1.mgmt_interface.ip_addr)
```

```
router_interfaces
ETH0 10.1.2.2
```

- In this example we could have defined *mgmt_interface* as a dictionary
- Use class instances where they fit logically, this depends on the design of your application
- In Python 3.8 there is a special class type – **data class**

https://realpython.com/python-data-classes/

# Demo

# Summary and next steps

- Summary

Classes

Class instances – Objects

Class properties/attributes

Class methods

OOP – Object-oriented programming

Next time

- SDKs – what are they, how to use them
- Building your own API server – intro to Flask