Advanced Python class

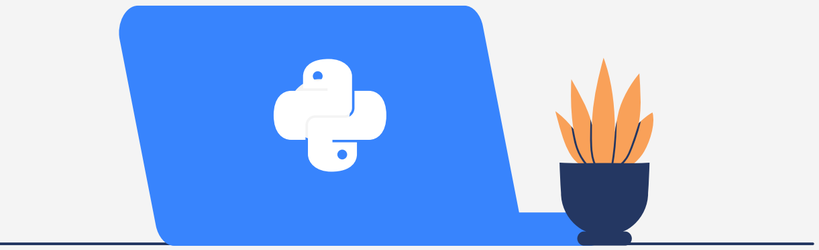
Cloud & DevOps Training Day 8

python

Advance

Topics

- Variable Scope
- Classes
- File Handling
- Using JSON
- Virtual Environment
- Advanced Function Topics
- Packages
- Directories Handling
- Request Module
- Flask & Django



@Sandip Das

Variable Scope: Global, Local and Nonlocal

Global Scope

Any variable created in the main body of the Python code and outside any function called global variable and scope of the variable is the global scope.

Note: We can create a global function from inside a function as well by using the **global** keyword.

Local Scope

Any variable created inside a function i.e. inside the local scope is called local scope.

Nonlocal Scope

Variables that are neither local nor global are of nonlocal scope.

Nonlocal variables are used in the nested functions as **nonlocal** keyword gets used to define Nonlocal variable

Example

```
x = "I am a Global Variable"
def someFunction():
  print("x inside function:", x)
someFunction()
print("x outside function:", x)
```

Usng global keyword

```
def mySampleFunction():
  global x
  x = 500
```

local scope

```
def someSimpleFunction():
 x = 500
 print("x inside function:", x)
someSimpleFunction()
```

nonlocal scope

```
def outerFunction():
  x = "I am local value"
  def innerFuntion():
   nonlocal x
   x = "I am non-local value"
   print("inner function value of x:", x)
 innerFuntion()
 print("outer function value of x:", x)
outerFunction()
```

Advanced Functions of Python

Documentation: docstring

Documentation string or in short "docstring" is used to describe what the function does.

Example

docstring

```
def greetPerson(name):
"""
   This function greets someone by using the name parameter
"""
   print("Hello, " + name)

greetPerson("Sandip")
```

Anonymous/Lambda Function

In Python, the function which doesn't have a defined name is called the Anonymous function.
In Python, anonymous functions are defined using the lambda keyword and a lambda function can take any number of arguments, but it can only have one expression.

Anonymous/Lambda Function

```
doSum = lambda a,b : a + b
print(doSum(10,20))
```

Recursive Function

A function can call another function, but in Python, a function can call itself. It's called - **Recursive Function**

Recursive Function

```
# factorial calculation using recursion
def calculateFactorial(x):
 if x == 1:
 return 1
 else:
 return (x * calculateFactorial(x-1))
num = 4
facorial_number = calculateFactorial(num)
print("Factorial of ", num, "is 1 * 2 * 3 * 4 = ", facorial_number)
```


@Sandip Das

Functions Advance Topics

Nested Function

A function defined inside another function is called nested function.

Closures

Python Closures are these **nested / inner** functions that are enclosed within the outer function. Closures can access variables that are present in the outer function scope.
It can access these variables even after the outer function has completed its execution

Decorator

A **decorator** is a design pattern in Python that allows us to **add new functionality to an existing code without modifying its structure.** The **decorator** acts as a wrapper.
It is also called **metaprogramming** because a part of the program tries to modify another part of the program during the compile time.

Use cases:

- Authorization in Python web frameworks such as Flask and Django
- Logging and execution time tracking

Example

Nested Function

```
def greet(name):
  def greetFirstName():
    # This is the nested function
    print("Hello ",name)
  greetFirstName()
greet("Sandip")
```

Closures

```
def greet(name):
#here name is nonlocal variable
  def greetFirstName():
    # This is the nested function
    print("Hello ",name)

  return greetFirstName

printFirstName = greet("Sandip")
printFirstName()
```

decorator

```
def formatGreet(func):
  def innerfunc(name):
    print("***************")
    func(name)
    print("***************")
  return innerfunc

@formatGreet
def greetFirstName(name):
 print("Hello ",name)
```


@Sandip Das

Python Classes

Why use Classes in Python?

One of the popular approaches of programming is by creating objects. This is known as Object-Oriented Programming (OOP). Python is an Object-Oriented Programming language, whose one of the key objectives of using classes is to utilize r**eusable code** as much as possible. This concept is also known as **DRY** (Don't Repeat Yourself).

Class

A Class is like a "blueprint" for creating objects.

Object

An object is an instantiation of a class. In simple terms an object is a collection of variables and functions that act on the provided data defined in the class.
Note: Whenever an object calls its method, the object itself is passed as the first argument, for example, we have used "self", but we can name this parameter anything as we like.

Constructors in Python

Class functions that begin with double underscore __ are called special functions, out of which the most used one is: __init__() function. All classes that have a function called __init__(), which is always executed when the class is being initiated.

Example

class

```
class MySimpleClass:
 x = 20
 def printValueOfX(self):
 print('Value of x is ', self.x)
```

Object

```
 c = MySimpleClass()
 c.printValueOfX()
```

Constructor

```
class person:
  def __init__(self, name, age):
    self.name = name
    self.age = age
  def greetPerson(self):
    print("Hello " + self.name)

p = person("John", 36)
p.greetPerson()
```

Python Class Inheritance

What is Inheritance in Python?

We can define a new class with little/no modification to the existing class. It allows us to define a class that inherits all the methods and properties from another class. The class from which being inherited from called parent/base class. The new class which inherits from the parent/base class is called the child/ derived class.

Example of Inheritance

```
class Patient(Person):
 def __init__(self, name, age, case):
 Person.__init__(self, name, age)
 case = case
 def print_case(self):
 print("Patient's case is " + self.case)

patientObject = Patient("John", "30",
"cold")
patientObject.greetPerson()
patientObject.print_case()
```

Super function

Python also has a super() function that will make the child class inherit all the methods and properties from its parent. Using super() function, we will not have to use the name of the parent element. It will automatically inherit the methods and properties from its parent.

super() function

```
class Patient(Person):
  def __init__(self, name, age, case):
    super().__init__(name, age)

newParient = Patient("John", "Doe", "Fever")
newParient.greetPerson()
```

@Sandip Das


@Sandip Das

Python Class Inheritance

Multiple Inheritance

If a Class derived from more than one base class, that is called multiple inheritance in python

Multi Level Inheritance

In Python a class can inherit from a derived class, it's called multi-level inheritance

Note: The **pass** statement is a null statement. The difference between a comment and a **pass** statement in Python is that while the interpreter ignores a comment entirely, **pass** is not ignored

Example

Multipl Inheritence

```
class SapleBase1:
 pass

class SampleBase2:
 pass

class MultiInheritenceExample(SapleBase1,
SampleBase2):
 pass

sample = MultiInheritenceExample()

print(sample.__class__.__bases__)
```

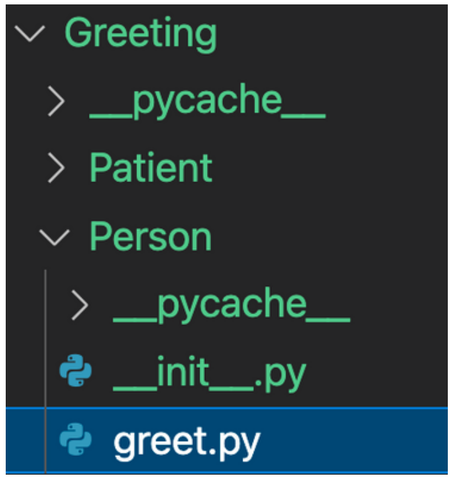
Multi Level Inheritence

```
class Level1:
 pass

class Level2(Level1):
 pass

class MultipleLevel(Level2):
 pass
multiLevelInheritenceObject = MultipleLevel()
print(MultipleLevel.__mro__)
print(MultipleLevel.__bases__)
```

Python Packages

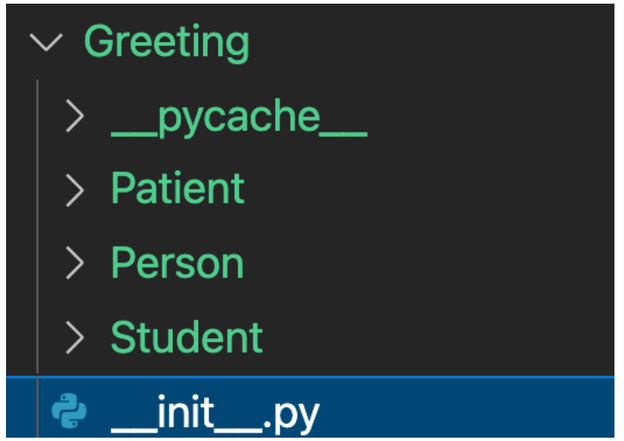
What is a Package in Python?

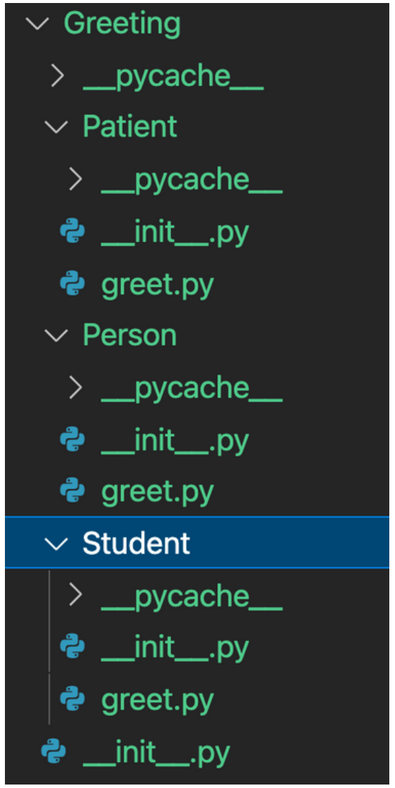
The package in Python in a simply a directory with Python files and a file with the name __init__ . py.
This means that every directory inside of the Python path, which contains a file named __init__ . py, will be treated as a package by Python.
A Python package can have sub-packages and modules, It's a well-organized hierarchy of directories for easier access.



Greet.py

```
def say_hello():
 print("Hello Person!")
```





File Handling In Python

What is a file ?

A File is a named location (object) on a disk to store data/information. File used to store data permanently on non-volatile memory such as Hard Disk (HD) / Solid State Drive (SSD) etc.

What File Operation Supported by Python ?

Python has several functions for creating, reading, updating, and deleting files.

What's the usual file handling flow?

1. Open a file
2. Read or write (do perform the intended operation)
3. Close the file

(Look at the examples below.)

Python use open() function to open a file and has the following modes:

"**r**" - Read - Default value. Opens a file for reading, error if the file does not exist

"**a**" - Append - Opens a file for appending, creates the file if it does not exist

"**w**" - Write - Opens a file for writing, creates the file if it does not exist

"**x**" - Create - Creates the specified file, returns an error if the file exists

"**t**" - Text - Default value. Text mode

"**b**" - Binary - Binary mode (e.g. images)

Example: Read

```
f = open("sample_file.txt", "r")
print(f.read())
f.close()
```

Example: Append

```
f = open("sample_file_2.txt", "a")
f.write("This is a line just overwritten the existing file\n")
f.close()
```

Example: Write

```
f = open("sample_file_2.txt", "w")
f.write("This is a line just overwritten the existing file if any\n")
f.close()
```

Example: Delete

```
import os
os.remove("sample_file_2.txt")
```

@Sandip Das


@Sandip Das

Python Directories

What are Directories in Python?

A directory or folder is a collection of files and folders(subdirectories).

How to Handle Directory / Sub-Directories in Python?

Python has the inbuilt os module which provides us with many useful methods/functions to work with directories, sub-directories, and files.

Example:

Get Current Working Directory:

```
import os
print(os.getcwd())
```

Change Directory:

```
import os
os.chdir('/var/logs')
print(os.getcwd())
```

List Directories and file:

```
import os
print(os.listdir())
```

Create new directory:

```
import os
os.mkdir('my_new_sample_directory')
```

Renaming file or directory:

```
import os
os.rename('my_new_sample_directory','my_renamed_sample_directory')
```

Create new directory:

```
import os
os.rmdir("my_renamed_sample_directory")import
import shutil
shutil.rmtree('directory_with_sub_directoris_andfiles')
```

Using JSON in Python

What is JSON?

JSON stands for JavaScript Object Notation, it's is a human-readable lightweight data-interchange format, a syntax for storing and exchanging data. It is commonly used for transmitting data in web applications.
In Simple words: **JSON** is a language-independent data format.
The **JSON** values can only be one of the six data types (**strings**, **numbers**, **objects**, **arrays**, B**oolean**, **null**)

How to use JSON in Python?

Python has a built-in package called **json**, which can be used to work with **JSON** data

Example:

Parsing JSON data

```
import json
person_json = '{ "name":"Sandip", "age":30, "city":"Kolkata"}'
# parse x:
person_object = json.loads(person_json)
print("Peron's age is: ",person_object["age"])
```

Converting Python Dictionary to JSON String

```
person_object["age"] = 45
new_person_json = json.dumps(person_object)
print("New Person JSON string: ", new_person_json)
print("Data type of JSON Dump",
type(json.dumps(new_person_json)))
```

@Sandip Das

Python Request Module

Using the **Requests** module we can make requests to any web page easily.
The HTTP request returns a Response Object with all the response data (content, encoding, status, etc).
Install requests module using python3 -m pip install requests or pip install -r requirements.txt

The request module support below methods:
get(url, params, args): Sends a GET request to the specified URL
head(url, args): Sends a HEAD request to the specified url
post(url, data, json, args): Sends a POST request to the specified URL
put(url, data, args): Sends a PUT request to the specified url
patch(url, data, args): Sends a PATCH request to the specified url
delete(url, args): Sends a DELETE request to the specified url
r**equest(method, url, args)**: Sends a request of the specified method to the specified url

Example:

```
import requests
x = requests.get('https://httpbin.org/get')
print(x.text)
```


@Sandip Das

Virtual Environment in Python

We can use a virtual environment to manage the dependencies for your project, both in development and in production.

Why is it needed ?

The more Python projects we have, the more likely it is that you need to work with different versions of Python libraries, or even Python itself. Newer versions of libraries for one project can break compatibility in another project.

Virtual environments are independent groups of Python libraries, one for each project. Packages installed for one project will not affect other projects or the operating system's packages.

How to create Virtual Environment

Python comes bundled with the **venv** module to create virtual environments.

Example:

```
mkdir my_sample_project
cd my_sample_project
python3 -m venv venv
```

To activate the virtual environment:

```
. venv/bin/activate
```


@Sandip Das

Up & Running With Flask

Flask is a web framework, it’s a Python module that lets us develop web applications easily. It has a small and easy-to-extend core. It’s also a microframework that doesn’t include an ORM (Object Relational Manager) or such features.
It does have many cool features like url routing, template engine, and it's a WSGI web app framework.

How To Install Flask:

Flask Installation steps:

Create a virtual environment

```
mkdir flask_app_example
cd flask_app_example
python3 -m venv venv
. venv/bin/activate
pip install Flask
```

Save dependencies in a separate file:

```
pip freeze > requirements.txt
```

How To Run Flask App:

```
python index.py
```

Example:

```
from flask import Flask
from flask import request
from flask import jsonify

app = Flask(__name__)

@app.route("/")
def home():
 return "Welcome.."

@app.route("/helloworld")
def helloworld():
 user = request.args.get('user')
 if(user):
 return "Hello " + user
 return "Hello Stranger"
```


@Sandip Das

Get Up & Running With Django

Django is a high-level Python web framework that encourages rapid development, clean, and pragmatic design. Built by experienced developers, it takes care of much of the hassle of web development, so you can focus on writing your app without needing to reinvent the wheel. It's also free and open source!

How To Install & Setup Django

Django Installation steps:
Create a virtual environment

```
mkdir django_app_example
cd django_app_example
python3 -m venv venv
. venv/bin/activate
pip install Django
```

save dependencies in a separate file:

```
pip freeze > requirements.txt
```

check installation:

```
python -m django --version
```

How To Run a Django Application

Creating a project

```
django-admin startproject mysimpleproject
```

to Run the app: run from mysimpleproject folder

```
python manage.py runserver
```

Run-on custom port:

```
python manage.py runserver PORT_NUMBER_HERE
```

How To Run Hello World app

Create New App:

```
python manage.py startapp helloWorld
```

then inside /mysimpleproject/helloWorld/views.py add below code:

```
from django.http import HttpResponse
def index(request):
 return HttpResponse("Hello, world")
```

Create file: /mysimpleproject/helloWorld/urls.py

```
from django.urls import path
from . import views

urlpatterns = [
 path('', views.index, name='index'),
]
```

in /mysimpleproject/mysimpleproject/urls.py add below code:

```
from django.contrib import admin
from django.urls import include, path
urlpatterns = [
 path('admin/', admin.site.urls),
 path('hellworld/', include('helloWorld.urls')),
 path('', include('helloWorld.urls')),
]
```

Assignment 1

Fetch multiple data from GitHub API via Request module, covert JSON data into a dictionary

GitHub API:

Look at the curl examples, then try to implement them in Python via request module

Ref: https://hevodata.com/learn/github-rest-apis/

Assignment 2

Create a Virtual Environment, then use Flask/Django to show a web page with good looking API

Basically, when visiting the home page, it should read an HTML file in the filesystem and in response send that file's contents to the browser

You can get the html themes from: https://www.free-css.com/free-css-templates