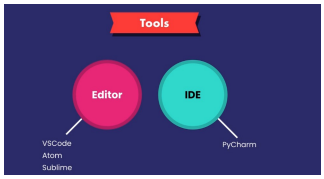
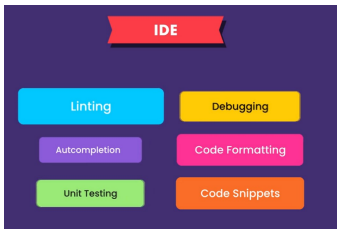
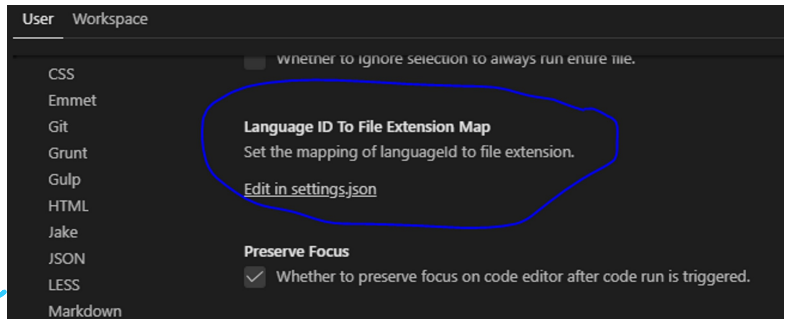
Editors and IDEs

22 August 2019 20:34



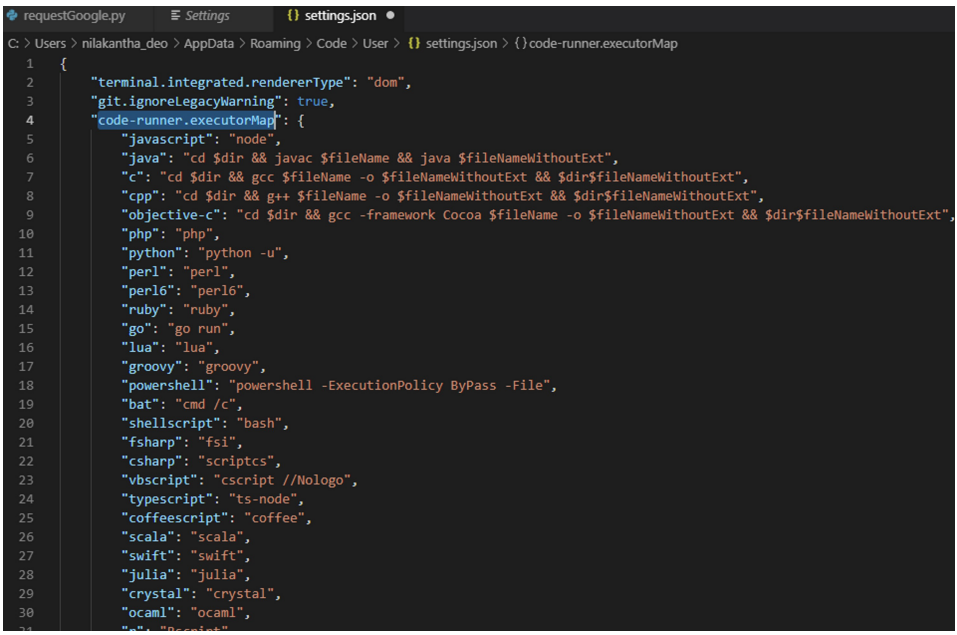




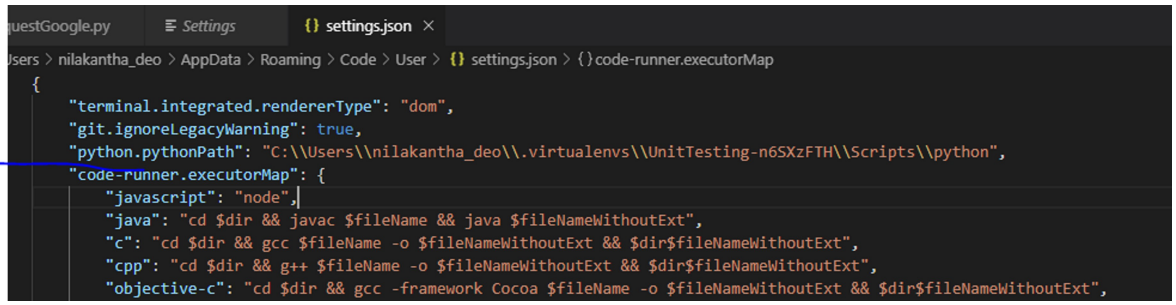
Install code runner extension
In user settings type **code-runner.executorMap to generate this json**

To use a particular path change the value of python to the venv path
Hint : use command **pipenv --venv**

Follow https://codewithmosh.com/courses/423295/lectures/8417860



Else add python.pythonpath



Deactivate to come out of an environment

Formatted string literals

21 August 2019 17:36

“formatted string literals,” f-strings are string literals that have an f at the beginning and curly braces containing expressions that will be replaced with their values.

From <https://realpython.com/python-f-strings/>

```
class Point:
    #class attributes
    defaultColor="red"
    def __init__(self,x,y):
        self.x=x
        self.y=y
    def draw(self):
        #print(f"Point ({self.x},{self.y})")
        print("self")
```

Function

20 August 2019 11:01

```
def increment(number:int,by:int=1):
    print(number+by)
increment(2)
```

List

20 August 2019 11:01

#append a value

```
letters=["A","b","c","a1"]
letters.append("d")
print(letters)
```

#insert at 0th location

```
letters.insert(0,"a1")
print(letters)
```

#remove without help of index

```
letters.remove("A")
print(letters)
```

#finding items

```
ind=letters.index("a1")
print("index of a1:",ind)
```

#finding number of occurence of an items

```
count=letters.count("a1")
print("count of a1: ",count)
```

#sorting a list

```
letters.sort()
print("sorted: ",letters)
```

#using method

```
letters.sort(reverse=True)
print("reverse sorted: ",letters)
```

#remove all from a list

```
letters.clear()
print(letters)
```

Touples

20 August 2019 12:48

```
items=[("i1",1),("i2",5),("i3",3),("i4",4)]

def sort_index(item):
   return item[1]

#get the keys
for i in items:
   print("keys:",sort_index(i))

#sorting the touples
items.sort(key=sort_index)
print("after sorting: ",items)
```

Mapped function

20 August 2019 13:08

```
items=[("i1",1),("i2",5),("i3",3),("i4",4)]
mappedArr=[];
for m in items:
   mappedArr.append(m[1])

print(mappedArr)
```

Lambda functions

20 August 2019 13:03

#Sorting

```
items=[("i1",1),("i2",5),("i3",3),("i4",4)]

#Get i such that i[1] using lambda

items.sort(key=lambda i:i[1])
print(items)
```

#mapped function steps

```
#create a lambda function with item such that taking 1st element
#map it to a list named items
#create a new list

print(list(map(lambda item:item[1],items)))
```

#filter functions steps

```
x=(filter(lambda item:item[1]>=3,items))
for xa in x:
    xa=list(x)
    print(xa)

#Or in single line
print(list(filter(lambda item:item[1]>=3,items)))
```

List Comprehensions

20 August 2019 14:34

```
items=[("a",1),("b",5),("c",3),("d",4)]

#With lambda function we can write as below
print("prices",list(map(lambda item:item[1],items)))

#Alternatively with list comprehension we can write as below
prices=[item[1] for item in items]
print("prices",prices)



list(filter(lambda item:item[1]>=3,items))

Prices=[item[1] for item in items if item[1]>3]
print("Prices",Prices)
```

Zip functions

20 August 2019 16:27

```
list1=[1,2,3]
list2=[4,5,6]

zippedList=zip(list1,list2)
#zippedList=zip("abc",list1,list2)
for item in zippedList:
    #item=list(zippedList)
    print(item)
```

Stack n Queue

20 August 2019 16:35

```
session=[]
session.append("url_1")
session.append("url_2")
firstElement=session.pop()
#session.pop()
print("firstElement",firstElement)
if not session:
   print("not empty",session)


#code for queue and deque
If not queue:
from collection import deque
```

Touples

20 August 2019 17:50

```
point1=(1,2)
point2=1,2
point3=1,
point4=()
point5=3*(1,2)
print(type(point1),type(point2),type(point3),type(point4))
print(point5)
point6=tuple([1,2])
if 1 in point6:
   print("yes",point6)
```

Output:

```
<class 'tuple'> <class 'tuple'> <class 'tuple'> <class 'tuple'>
(1, 2, 1, 2, 1, 2)
yes (1, 2)
```

From <https://www.onlinegdb.com/>

Swap

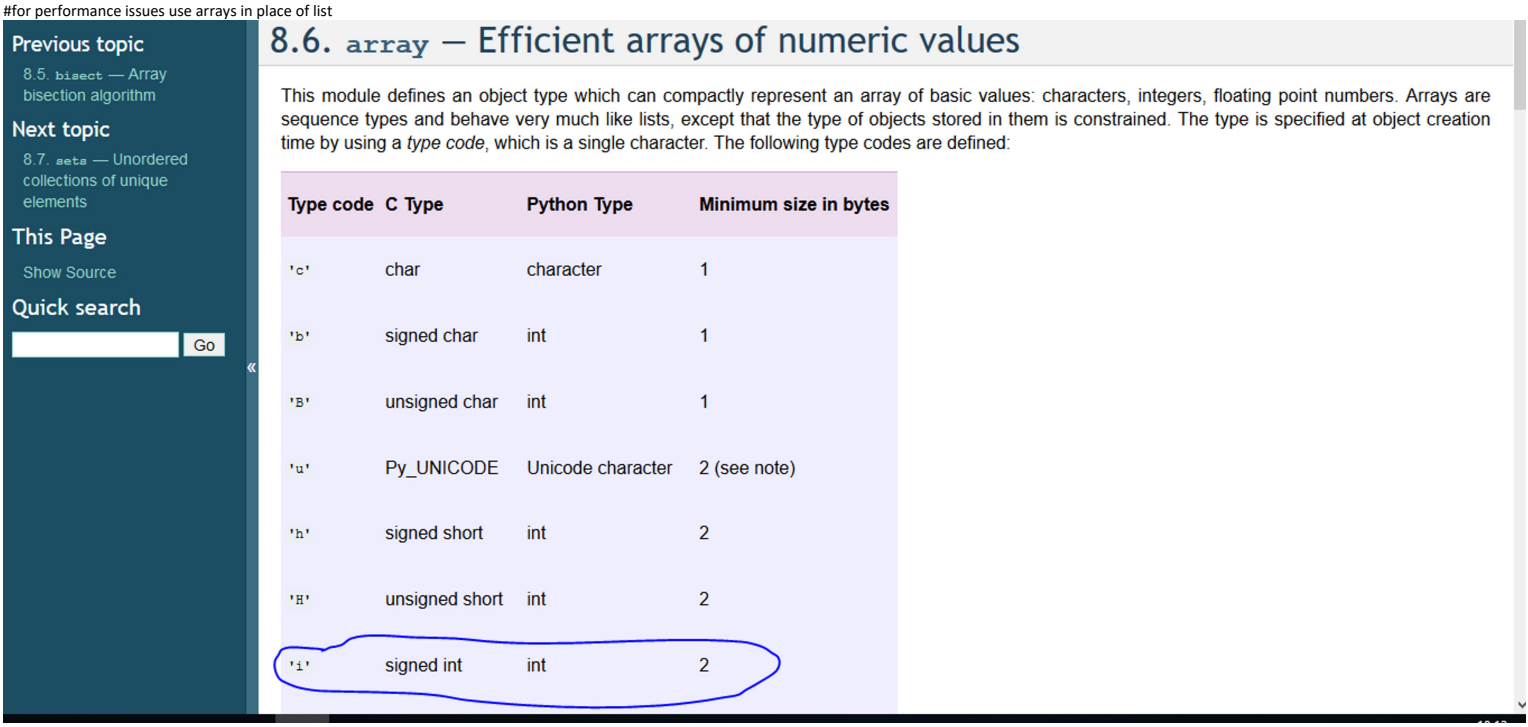
20 August 2019 17:57

```
x=10
y=11
x,y=(y,x)
print(x,y)
```

Arrays

20 August 2019 17:58

#for performance issues use arrays in place of list



Previous topic

8.5. bisect — Array bisection algorithm

Next topic

8.7. sets — Unordered collections of unique elements

This Page

Show Source

Quick search

Go

8.6. array — Efficient arrays of numeric values

This module defines an object type which can compactly represent an array of basic values: characters, integers, floating point numbers. Arrays are sequence types and behave very much like lists, except that the type of objects stored in them is constrained. The type is specified at object creation time by using a *type code*, which is a single character. The following type codes are defined:

Type code	C Type	Python Type	Minimum size in bytes
'c'	char	character	1
'b'	signed char	int	1
'B'	unsigned char	int	1
'u'	Py_UNICODE	Unicode character	2 (see note)
'h'	signed short	int	2
'H'	unsigned short	int	2
'i'	signed int	int	2

#all the objects are same types

```
from array import array
numbers =array("i",[2,3])
numbers.append(3)
print("numbers",numbers)
```

Output:
numbers: array('i', [2, 3, 3])

From <https://www.onlinegdb.com/>

sets

20 August 2019 18:15

#collection with no duplicates

```
n=[1,1,2,3,3,4,4,5]
testSet=set(n)
print(testSet)
```

Output

{1, 2, 3, 4, 5}

From <https://www.onlinegdb.com/>

From <https://docs.python.org/3/tutorial/datastructures.html>

#Union ,check the parenthesis and curly braces

```
#(a | b     # letters in a or b or both)
firstSet={1,2,3,4,4}
secSet={4,5,5,6,6,7}
resUnion=(firstSet|secSet)
print("resUnion",resUnion)
```

#Intersection

```
#(a & b    # letters in both a and b
resIntersection=(firstSet&secSet)
print("resIntersection",resIntersection)
```

#Difference

```
#a - b      # letters in a but not in b
resDifference=(firstSet-secSet)
print("resDifference",resDifference)
```

#Symmetric difference

```
#a ^ b     # letters in a or b but not both
resSym=(firstSet^secSet)
print("resSym",resSym)
```

Dictionary(dict keyword)

20 August 2019 20:00

```
point={"x":1,"y":2}
point2={}
point3=dict(x=1,y=2)
print("point3: ",point3)
print("index: ",point3["x"])
print("index: ",point3.get("x"))

for key in point3:
   print("key:",key,"val:",point3[key])

for x,y in point.items():
   print(x,y)
```

Output:

```
point3:  {'y': 2, 'x': 1}
index:  1
index:  1
key: y val: 2
key: x val: 1
y 2
x 1
```

From <https://www.onlinegdb.com/>

Note: numeric keyword not supported

Dictionary comprehension

...

Dictionary comprehension(not complete)

20 August 2019 20:02

Unpacking Operations(like spread ...)(not complete)

20 August 2019 20:03

Exceptions(not complete...)

20 August 2019 20:04

Class vs instance method and decorators

21 August 2019 18:41

```
class Point:
   def __init__(self,x,y):
      self.x=x
      self.y=y

   #decorate the classmethod
   @classmethod
   def zero(cls):  #pass it cls
      return cls(0,0) #assign values and return the cls object

point=Point.zero() #no need to assign again and again
print(point.x,point.y)
```

Classes n Constructors

21 August 2019 15:53

#Class
Every object in python is created using a class
#define should be passed with the **self**

```
class Point:
    def draw(self):
        print("draw")

point=Point()
point.draw()
```

#Constructors
#self is a reference to the current object
#__init__ is a magic method

```
class Point:
    #class level attributes below can be accessed across all objects
    defaultColor="red"
    def __init__(self,x,y):
        self.x=x
        self.y=y
    def draw(self):
        #print(f"Point ({self.x},{self.y})")
        print("self")

point=Point(1,2)
#accessing class attributes
print(point.defaultColor)
print(Point.defaultColor)

point.draw()

anotherPoint=Point(5,2)
point.draw()
```

Magic methods

21 August 2019 18:44

Have __ and __ at the beginning and end

Search for rszalski.github.io

```
class Point:
   def __init__(self,x,y):
      self.x=x
      self.y=y
   def __str__(self):
      return f"({self.x},{self.y})"


point=Point(1,2)

print(str(point))
```

Comparing objects(magic methods) and addition

21 August 2019 18:51

```
class Point:
   def __init__(self,x,y):
      self.x=x
      self.y=y


point=Point(1,2)
otherPoint=Point(1,2)
print(point==otherPoint)

#output==false (because it compares reference of the object)

Check for magic method __eq__
class Point:
   def __init__(self,x,y):
      self.x=x
      self.y=y

      #Defines behavior for the equality operator, ==.
   def __eq__(self, other):
      return self.x==other.x and self.y==other.y

     #Defines behavior for the equality operator, >

   def __gt__(self, other):
      return self.x>other.x and self.y>other.y


point=Point(3,2)
otherPoint=Point(1,1)
print(point>otherPoint)


#for addition try this

def __add__(self, other):
      return Point(self.x+other.x , self.y+other.y)


point=Point(3,2)
otherPoint=Point(1,1)
P=point+otherPoint
print(P.x,P.y)
```

Custom container(a dictionary)

21 August 2019 19:48

```
class TagCloud:
    def __init__(self):
        self.tags={}

    def add(self,tag):
        #get a tag with key tag and assign to 0 if value is null else increament by 1
        #set it to a new value at the left side
        #self.tags[tag]=self.tags.get(tag,0)+1  here case sensitive
        self.tags[tag.lower()]=self.tags.get(tag.lower(),0)+1

    #def __getitem__(self,key):
    def __getitem__(self,tag):
        return self.tags.get(tag.lower(),0)

    #def __setitem__(self,key,value)
    def __setitem__(self,tag,count):
        self.tags[tag.lower()]=count

    def __len__(self):
        return len(self.tags)

    def __iter__(self):
        return iter(self.tags)

cloud=TagCloud()
cloud.add("Pu")
cloud.add("pu")
cloud.add("pu")

print(cloud.tags)

#uses __getitem__ method
count=cloud["pu"]
print("count:",count)


#uses __setitem__ method
cloud["pu"]=114
print("set new value:",cloud.tags)


#uses __len__ method
length=len(cloud)
print("length:",length)

#uses __iter__ method
itera=iter(cloud)
```

```
print("iter:",itera)
```

Output:

```
{'pu': 3}
count: 3
set new value: {'pu': 114}
length: 1
iter: <dict_keyiterator object at 0x7f58a7bbf728>
```

From <https://www.onlinegdb.com/online_python_compiler#editor_1>

Properties(getters and setters)(not documented)

21 August 2019 20:11

Inheritance

21 August 2019 20:25

```
class Animal:
   def __init__(self):
      self.age=2
   def eat(self):
      print("eats")

class Mamal(Animal):
   def walk(self):
      print("walks")
class Fish(Animal):
   def swim(self):
      print("swims")

m=Mamal()
m.eat()
m.walk()
print(m.age)

f=Fish()
f.eat()
f.swim()
print(f.age)
```

Output:
eats
walks
2
eats
swims
2

From <https://www.onlinegdb.com/online_python_compiler#editor_1>

Multilevel inheritance(avoid)

app.py

```
1  class Animal:
2      def eat(self):
3          print("eat")
4
5
6  class Bird(Animal):
7      def fly(self):
8          print("fly")
9
10 Employee - Person - LivingCreature - Thing
```

Multiple inheritance

The object class

21 August 2019 20:34

O=object()
isinstance
issubclass()

Method Overriding

22 August 2019 12:04

Overriding a method of the base class

```
class Animal:
   def __init__(self):
      self.age=1

class Fish(Animal):
   def __init__(self):
      super().__init__()
      self.weight=3

   def swim(self):
      print("swims")

class Mammal:

   def walk(self):
      print("walks")



f=Fish()
f.swim()

print(f.age)

print(f.weight)

m=Mammal()
m.walk()
```

17-22 (not completed)

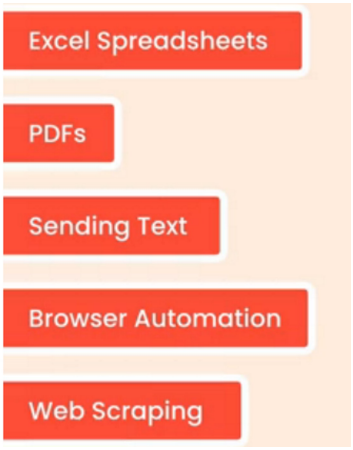
22 August 2019 13:04

Modules

22 August 2019 13:11

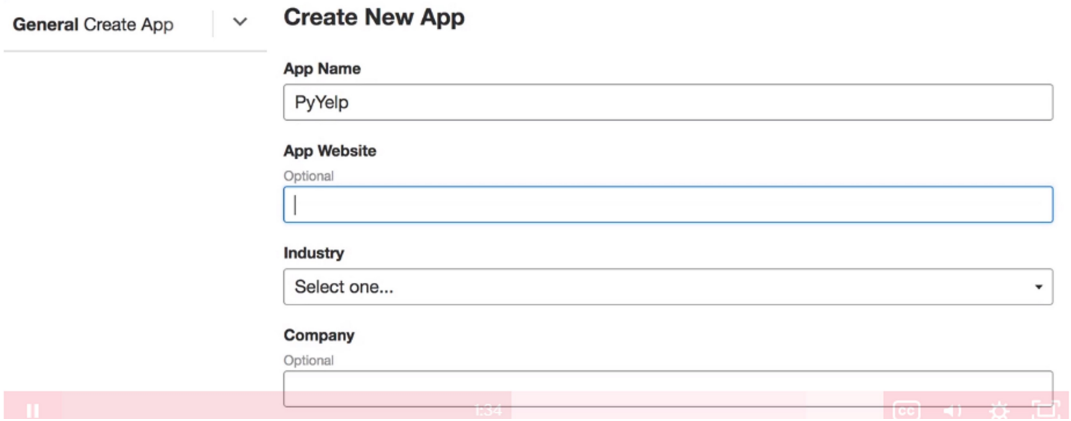
Popular Packages yelp,numpy,webscraping

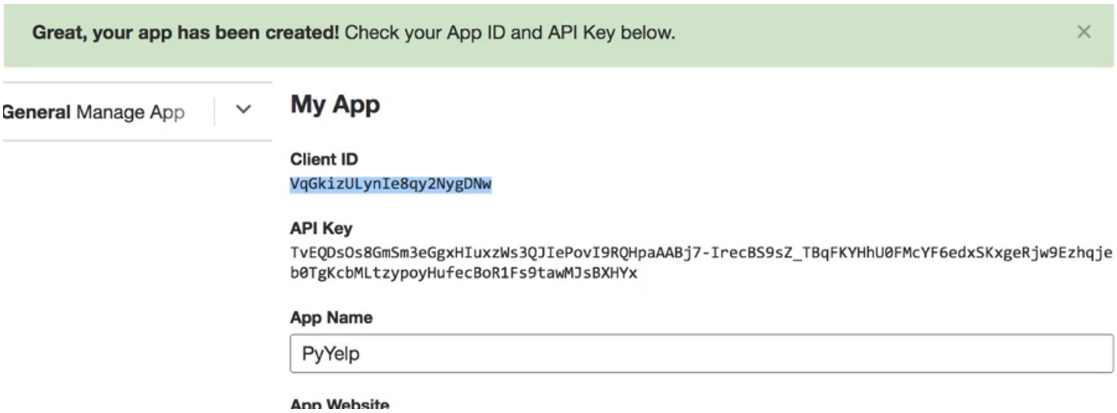
22 August 2019 15:18



YELP
https://www.yelp.com/developers
Yelp fusion rest API for yelp

Create App

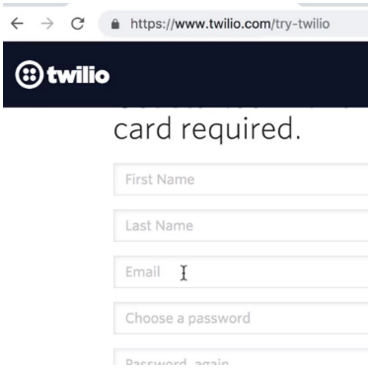




Authentication guide

Hiding Api keys

Sending text message
Twilio



Web scraping
Beautifulsoup4
Request

Python Package Index(pypi) and pip env

22 August 2019 17:56

Pypi.org
pip list to see all packages

Pip to install a package
https://pip.pypa.io/en/stable/installing/
To upgrade follow this command

python -m pip install --upgrade pip --user

To install a package such as requests follow this(only bold part works ,if not use whole command)

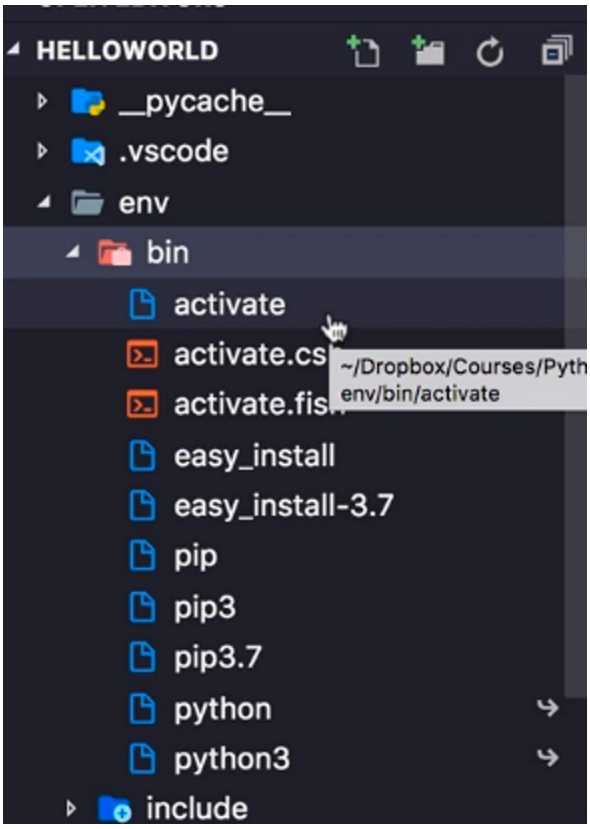
python -m **pip install requests** --user

pip install requests==2.9.0
Wildcard style
pip install requests==2.9.*
Latest package
pip install requests~=2.9.*

To uninstall
Pip uninstall request==2.9.0

Environment management

Create an environment
python -m venv env
Activate it
source env\bin\activate



After activation

To deactivate
deactivate simply

Using Alternatively pipenv

pip install pipenv

All virtuall envs
pipenv --venv

Install virtual envs for this project in a folder location
pipenv install

Activate virtual environmnts
pipenv shell

Install packages for the virtual env with **pip install <package_name>**

Virtual environment in vscode>>check Editors and IDEs

pipenv install --ignore-pipfile

Managing dependencies

https://codewithmosh.com/courses/python-programming-course-developers/lectures/8417862

pipenv graph

pipenv update --outdated

NumPy

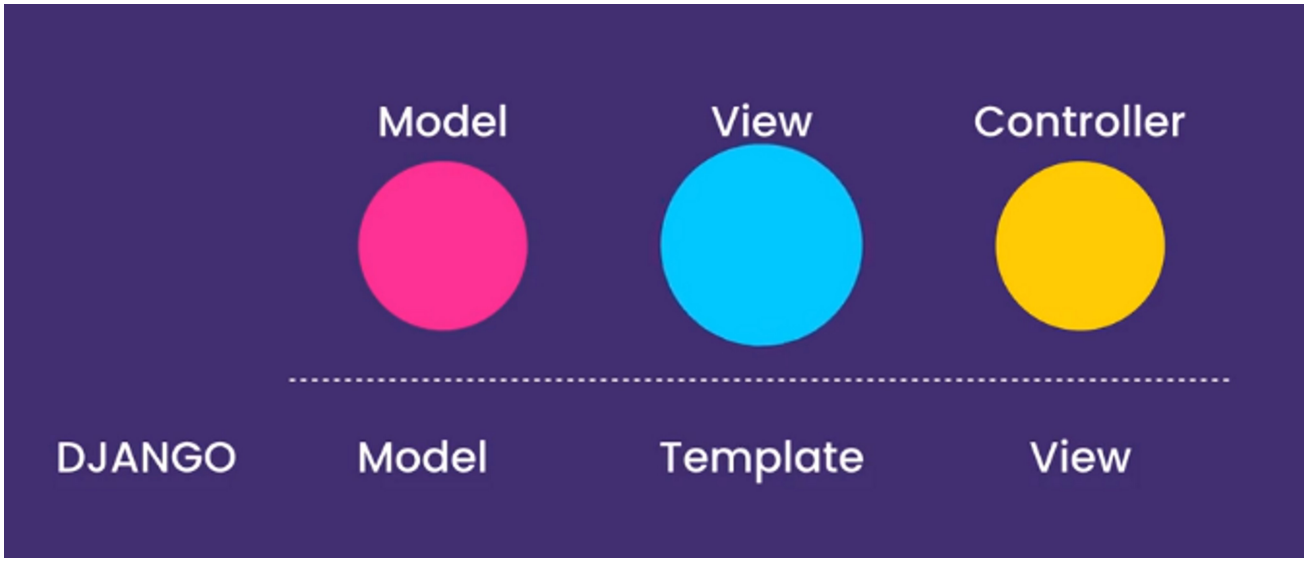
23 August 2019 12:37

Django

23 August 2019 14:54

mkdir DjangoProject
cd DjangoProject

pipenv install Django



django-admin startproject vidly .
To create a project vidly

python .\manage.py runserver

python manage.py startapp movies
Create details about movies in vidly app
python manage.py run server

Views.py in movies
----
from django.shortcuts import render
from django.http import HttpResponse

Create your views here.
def index(requests):
return HttpResponse("Hello Django man")

Add a file urls.py
----
from django.urls import path
from . import views

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

In vidly urls.py file add changes

```
from django.contrib import admin
from django.urls import path,include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('views/', include('movies.urls')),
]
```

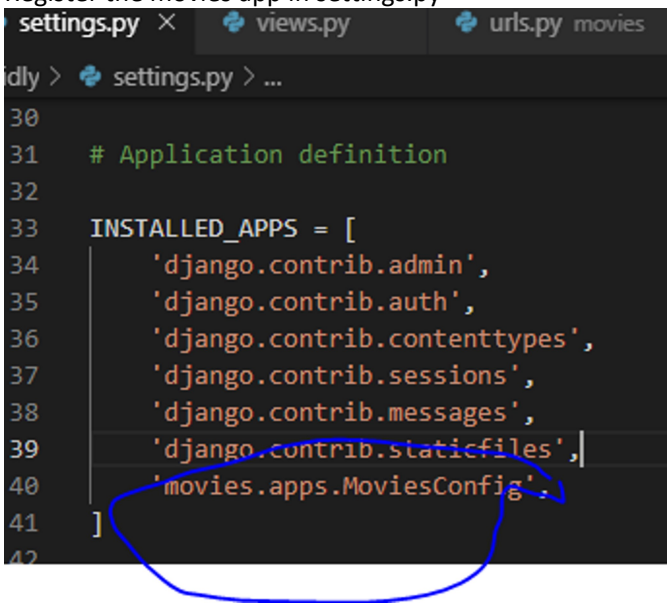
In models.py in movies folder

```
from django.db import models

# Create your models here.
# models has a class Model
class Genre(models.Model):
    name = models.CharField(max_length=255)
class Movie(models.Model):
    title=models.CharField(max_length=255)
    release_year=models.IntegerField()
    number_in_stock=models.IntegerField()
    daily_rate=models.FloatField()
    #each movie needs to be associated with a genre(if a genre is deleted all the movies associated are deleted)
    genre=models.ForeignKey(Genre,on_delete=models.CASCADE)
```

Install sqlite

Register the movies app in settings.py



Run the following

python manage.py makemigrations

Python manage.py migrate

Changing the model (add a new field date_created)

#adding a new field date_created
date_created=models.DateTimeField(default=timezone.now)

To add timezone import and use it on the top of the file as below

from django.utils import timezone

What sql statements are generated after each migration?

python .\manage.py **sqlmigrate** movies **0001**

sqlmigrate is the keyword and the migration file name is 0001_initial.py

ADMIN (administrator panel)

Run the server with
python manage.py runserver

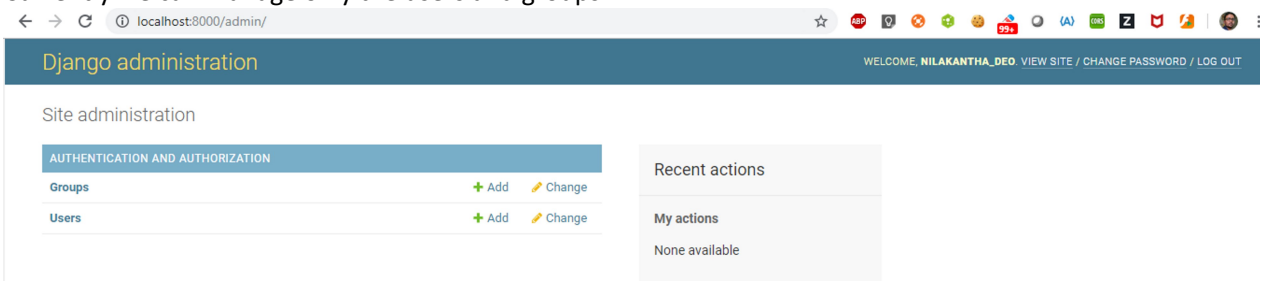
Create another terminal in vscode and run this command below to create a superuser

python .\manage.py createsuperuser

If error start the env command with

pipenv shell

Currently we can manage only the users and groups



To manage the models
We need to import the models in the admin.py file and register them too

from .models import Genre,Movie

# Register your models here.
admin.site.register(Genre)
admin.site.register(Movie)

But it shows as objects in the admin site
To see the exact name we need to change it in the models.py file

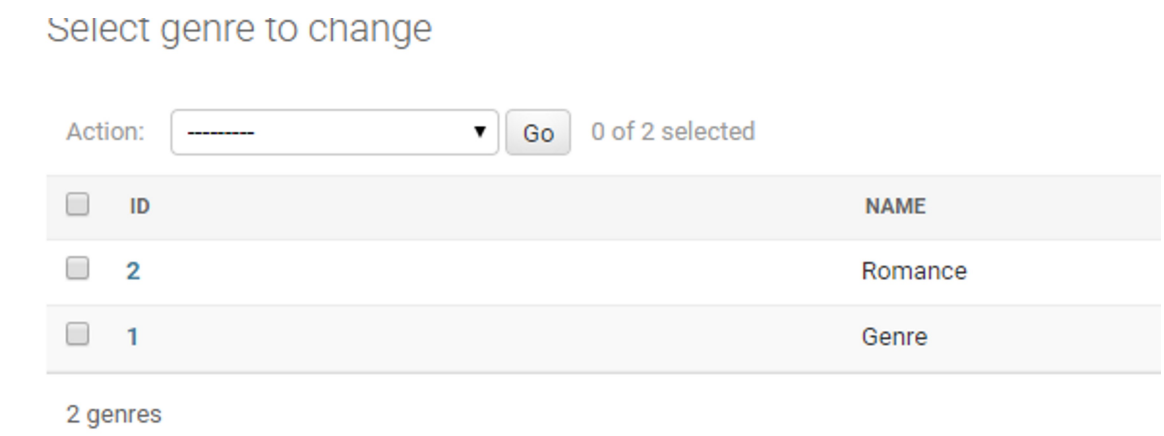
class Genre(models.Model):

```
    name = models.CharField(max_length=255)
    def __str__(self):
        return self.name
```

To show the ids along with the name
Create a class in the admin.py file and pass the class name in the register method

```
class GenreAdmin(admin.ModelAdmin):
    list_display=('id','name')

# Register your models here.
admin.site.register(Genre,GenreAdmin)
admin.site.register(Movie)
```



To remove a field from the model in admin

```
class MovieAdmin(admin.ModelAdmin):
    exclude=('date_created',)

    list_display=('id','title')

# Register your models here.
admin.site.register(Genre,GenreAdmin)
admin.site.register(Movie,MovieAdmin)
```

Note that the date_created is excluded and a comma is left at the end.Else it will be treated as a string wrapped in parenthesis .

Modifying the view

There is abstraction to work with databases in Django

```
# Create your views here.
def index(request):
    #SELECT * FROM movies_movie WHERE release_year =1998; is equivalent to the following
    #Movie.object.filter(release_year=1998)

    #SELECT * FROM movies_movie WHERE id =1; is equivalent to the following
    #Movie.object.get(id=1)
```

```
# SELECT * FROM movies_movie; is equivalent to the following API
movies = Movie.objects.all()

#Create a list taking titles from movie and join each elements with a ,
allMovies='  ,  '.join([m.title for m in movies])

return HttpResponse(allMovies)
```

Templates in Django

In views.py file

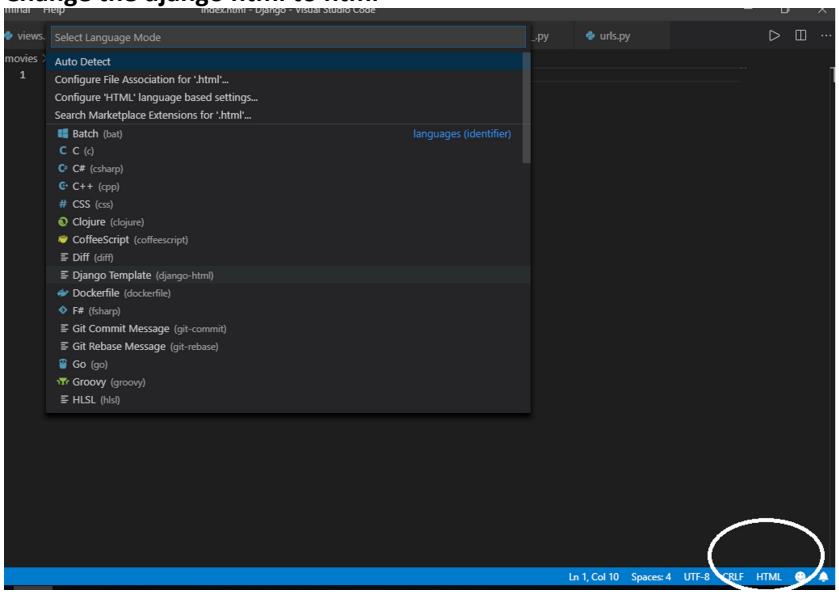
```
moviesObject = Movie.objects.all()
  allMovies='  and  '.join([m.title for m in moviesObject])

  #to render it in a template
  #pass the request and the name of the template file
  #create a dictionary and pass a key as movies and values as the moviesObject
  # return it

  return render(request,'index.html',{'movies':moviesObject})
```

Create a folder templates and add a file index.html

Change the django html to html



Use zencoding for creating htmlhttps://www.smashingmagazine.com/2009/11/zen-coding-a-new-way-to-write-html-code/

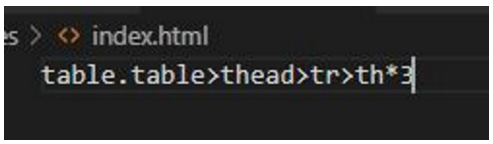


Table.table says table is a class of table

```
<table class="table">
    <thead>
        <tr>
            <th>Title</th>
            <th>Genre</th>
            <th>Release Year</th>
        </tr>
    </thead>
    tbody>tr>td*3
</table>
```

Use curly braces and % to iterate the object *movies* defined earlier
For that change html back to django html

And move td inside for

```
    <tbody>
        {%{% for movie in movies %}

        {% endfor %}%}
        <tr>
            <td></td>
            <td></td>
            <td></td>
        </tr>
    </tbody>
/table>
```

The above looks as below

{% for movie in movies %}
<tr>
<td>{{movie.title}}</td>
<td>{{movie.genre}}</td>
<td>{{movie.release_year}}</td>
</tr>
{% endfor %}

Pylint complains

```
# SELECT * FROM movies_movie; is equivalent t
moviesObject = Movie.objects.all()
allMovies='  and  '.join([m.title for m in mo
```

Install the following
pipenv install pylint-django
Then create a file .pylintrc in the root folder and write the following
load-plugins=pylint-django

For best practices put the index.html inside another folder (else some index file of another namespace

will be loaded first Lecture 11)
Say movies/index.html
So in views.py it becomes
return render(request,'**movies/index.html**',{'movies':moviesObject})

Bootstrap

Go to https://getbootstrap.com/docs/4.3/getting-started/introduction/**And copy the code snippet**

Create a base template file base.html inside the movies/template folder and paste the above code

Add these two lines after body using Django template and block keyword and adding **content**

```
<body>
  {% block content %}
  {% endblock  %}
```

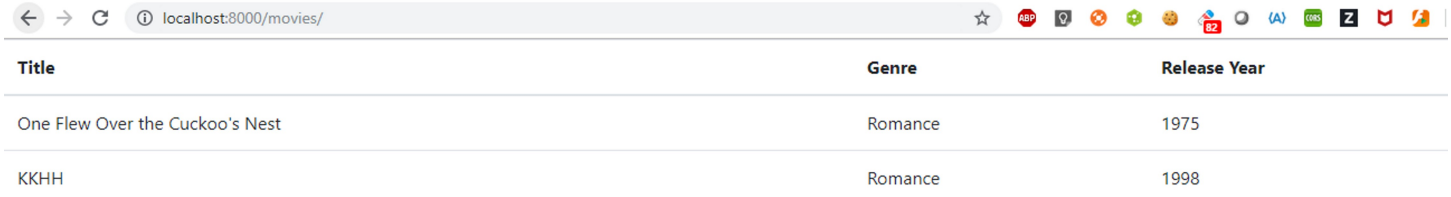
This base.html should be used as a base and should be extended in other html files

Ex: in index.html
{% extends "movies/base.html" %}

And place all the tags inside the block command

```
{% block content  %}
<table>
... All tags
...
</table>
{% endblock  %}
```

Output



Customizing the layout

Add a navbar in the base.html

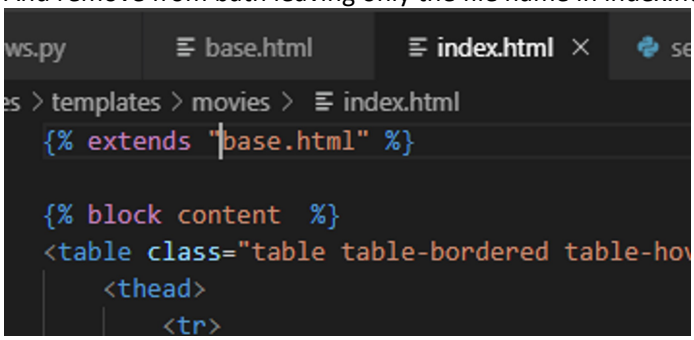
```
<body>
  <nav class="navbar navbar-light bg-light">
    <a class="navbar-brand" href="#">Vidly</a>
  </nav>
```

Sharing a base template across

Context processors inside vidly>settings.py keeps track of
Create a templates folder in the root dir and move base.html inside it

In DIRS add the path of the templates folder
'DIRS': [os.path.join(BASE_DIR,'templates')],

And remove from bath leaving only the file name in index.html as below



Url Parameters

To make parameterised change in urlpatterns in url.py
in vidly>urls.py we have the following

'DIRS': [os.path.join(BASE_DIR,'templates')],urlpatterns = [
path('admin/', admin.site.urls),
path('movies/', include('movies.urls')),
]

The bold line above says that whenever the endpoint is movies/ we have to look for movies.urls file
So in urls.py we have to add the following

urlpatterns = [
path('',views.index,name='movies_index'),
path('<movie_id>',views.detail,name='movies_detail')
]

Create an id **movie_id** and pass it in a method **detail** in **views.py**. Give a namespace **movies_detail**. Also change index to **movies_index** for better maintenance.

Getting a single object (/movies/1)

Create a method in the views.py as below

def detail(request,movie_id):
movieDetailsObject=Movie.objects.get(id=movie_id)
return render(request,'movies/details.html',{'movies_details':movieDetailsObject})
#return HttpResponse(movie_id)

The bold line says create a template details.html and a dictionary with a key details and assign it to value DetailsObject and return it

Create a file details.html inside the movies folder and add the below code (dl is description list)

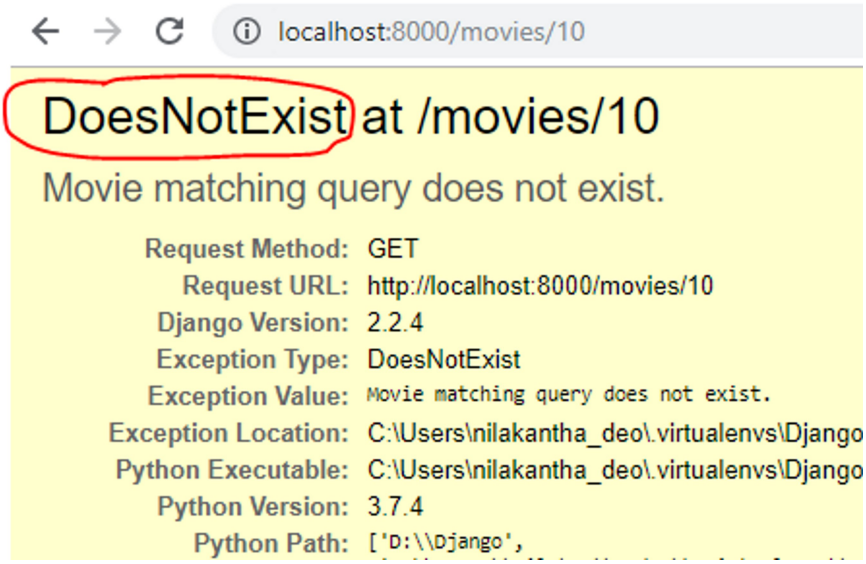
{% extends 'base.html' %}
{% block content %}
<dl>
<dt>Title</dt>
<dd>{{movies_details.title}}</dd>

```
  <dt>Year</dt>
  <dd>{{movies_details.release_year}}</dd>
  <dt>Stock</dt>
  <dd>{{movies_details.number_in_stock}}</dd>
</dl>

{% endblock  %}
```

Raising 404 error

If we pass an invalid endpoint say 10 we get



To raise a 404 exception we need to import it from Django

from django.http import HttpResponse,Http404

```
def detail(request,movie_id):
  try:
    movieDetailsObject=Movie.objects.get(id=movie_id)
    return render(request,'movies/details.html',{'details':movieDetailsObject})
    #return HttpResponse(movie_id)

  except Movie.DoesNotExist:
    raise Http404()
```

_____

Alternatively we can import **get_object_or_404**

from django.shortcuts import render,get_object_or_404

movieDetailsObject=get_object_or_404(Movie,id=movie_id)

_____

Referencing URL

In index.html add the following

```
<tr>
        <td>
          <a href="/movies/{{movie.id}}">
            {{movie.title}}
          </a>
        </td>
```

Important: For better practices refer lecture 18

Or we can use url as below
<a href="{% url 'movies_detail' movie.id %}">
In place of <a href="/movies/{{movie.id}}"> (see above)

Now in the urls.py we usually give namespace movies_details .
Rather we can give the app name as movie and remove movies_ from other places as below

from django.urls import path
from . import views

app_name='movies'

urlpatterns = [
path('',views.index,name='index'),
path('<movie_id>',views.detail,name='detail')
]

And in index.html instead of underline we can use :(so that django knows its the app movies with details)
<a href="{% url 'movies:detail' movie.id %}">