# Python

## Array & If/Else Statement

movies = ["Gone in 60 seconds", "Madagascar", "Sweet Dream"]

**for movie in movies:**

**if(movie == 'Sweet Dream'):**

print("Must See: " + movie)

**else:**

print(movie)

## Functions

**import math**

***def* circle(*radius*):**

return math.pi \* math.pow(radius, 2)

print("Area of the circle: ", circle(5));

## Creating Distribution

**python setup.py sdist**

**python setup.py install**

### Sample setup.py

**from distutils.core import setup**

setup(

*name* = 'nester',

*version* = '1.0.0',

*py\_modules* = ['nester'],

*python* = 'hfpython',

*author\_email* = 'hfpython@headfirstlabs.com',

*url* = 'http://www.headfirstlabs.com',

*description* = 'A simple printer of nested lists'

)

### Sample nester.py

*def* print\_lol(*the\_list*, *indent* = False, *level* = 0):

for each\_item in the\_list:

if isinstance(each\_item, *list*):

print\_lol(each\_item, indent, level + 1)

else:

if indent:

for tab\_stop in **range(level)**:

**print("\t", *end*='')**

print(each\_item)

## File Input/Output

### random.txt

Tom: What are you eating?

I: I ain't eating anthing!

(pause)

Dick: May be you are up to something

Mary: Hell no

(pause)

Satan: Welcome to hell?

Harry: Netflix or Chill?

### read.py

**import os**

**try:**

if **os.path.exists('random.txt'):**

data = **open('random.txt')**

**# print(data.readline(), end='')**

for each\_line in data:

try:

if not each\_line.**find(":")** == -1:

(role, line\_spoken) = each\_line.**split(":", 1)**

print(role, *end*='')

print(' said: ', *end*='')

print(line\_spoken, *end*='')

except *ValueError*:

pass

data.close()

else:

print('The data file is missing!')

**except *IOError*:**

print('The data file is missing!')

### write.py

try:

write = **open('random\_write.txt', 'w')**

print('I am just writing bro!', ***file*=write**)

except *IOError*:

print("File Error")

finally:

write.close()

## Pickle

### write.py

**import pickle**

**with open**('mydata.pickle', **'wb'**) **as mysaveddata**:

**pickle.dump**([1, 2, 'three'], mysaveddata)

with open('mydata.pickle', '**rb**') as myrestoredata:

retrieve = **pickle.load**(myrestoredata)

print(retrieve)

## Sort

**In place sorting** takes our data, arranges it in the order we specify, and then *replaces our original data with the sorted version*. **sort()**

**Copied sorting** take our data, arranges it in the order we specify and then *returns a sorted copy of our original data*. **sorted()**

## List Comprehension

clean\_james = []

for each\_record in james:

clean\_james.append(sanitize(each\_record))

clean\_james = **[sanitize(each\_record) for each\_record in james]**

## Set

Removes the duplicates

**set(james)**

## List Slice

james = [5, 6, 9, 10, 23, 56, 2]

**james[0:3]**

### data.py

def sanitize(string\_data):

if '-' in string\_data:

splitter = '-'

elif ':' in string\_data:

splitter = ':'

else:

return string\_data

(mins, secs) = string\_data.split(splitter)

return(mins + '.' + secs)

def get\_data(filename):

try:

with open(filename) as data:

data = data.readline()

return data.strip().split(',')

except IOError as ioerr:

print("Pickle Error: " + str(ioerr))

return(None)

james = get\_data('james.txt')

julie = get\_data('julie.txt')

mikey = get\_data('mikey.txt')

sarah = get\_data('sarah.txt')

**james = sorted(set([sanitize(each\_record) for each\_record in james]))**

**julie = sorted(set([sanitize(each\_record) for each\_record in julie]))**

**mikey = sorted(set([sanitize(each\_record) for each\_record in mikey]))**

**sarah = sorted(set([sanitize(each\_record) for each\_record in sarah]))**

print(james[0:3])

print(julie[0:3])

print(mikey[0:3])

print(sarah[0:3])

## Objects

### objects.py

*def* sanitize(*each\_record*):

if '-' in each\_record:

splitter = '-'

elif ':' in each\_record:

splitter = ':'

else:

return each\_record

(mins, secs) = each\_record.split(splitter)

return (mins + "." + secs)

*def* get\_data(*filename*):

try:

with open(filename) as data:

data = data.readline()

data = data.strip().split(',')

return (**{'Name': data.pop(0),**

**'DOB': data.pop(0),**

**'Times': *str*(sorted(*set*([each\_record for each\_record in data]))[0:3])}**)

except *IOError* as ioerr:

print("IOError: " + *str*(ioerr))

return(None)

photon = get\_data('photon.txt')

print('Name: ' + photon['Name'] + ' :: DOB: ' + photon['DOB'] + ' :: Fastest Times:' + photon['Times'])

## Class

### athlete.py

*def* sanitize(*data*):

if '-' in data:

splitter = '-'

elif ':' in data:

splitter = ':'

else:

return data

(mins, secs) = data.split(splitter)

return (mins + "." + secs)

*def* get\_data(*filename*):

try:

with open(filename) as data:

data = data.readline()

data = data.strip().split(',')

return ({"name": data.pop(0), "dob": data.pop(0), "times": data})

except *IOError* as ioerr:

print("IOError: " + ioerr)

return(None)

james = get\_data('james.txt')

julie = get\_data('julie.txt')

mikey = get\_data('mikey.txt')

sarah = get\_data('sarah.txt')

# print(james['name'] + "::" + james['dob'] + "::" + str(james['times']))

***class* Athlete:**

***def* \_\_init\_\_(*self*, *athlete*):**

**self.name = athlete['name']**

**self.dob = athlete['dob']**

**self.times = athlete['times']**

***def* best(*self*):**

**return(sorted(*set*([sanitize(each\_record) for each\_record in self.times]))[0:3])**

**james = Athlete(james)**

print("Athlete Name: " + james.name + ", DOB: " + james.dob + ", Top 3:" + *str*(james.best()))

### athelelist.py

***class* AthleteList(*list*):**

*def* \_\_init\_\_(*self*, *athlete*):

self.name = athlete['name']

self.dob = athlete['dob']

self.times = athlete['times']

*def* best(*self*):

return(sorted(*set*([sanitize(each\_record) for each\_record in self.times]))[0:3])

*def* sanitize(*data*):

if '-' in data:

splitter = '-'

elif ':' in data:

splitter = ':'

else:

return data

(mins, secs) = data.split(splitter)

return (mins + "." + secs)

*def* get\_data(*filename*):

try:

with open(filename) as data:

data = data.readline()

data = data.strip().split(',')

data = {"name": data.pop(0), "dob": data.pop(0), "times": data}

**return AthleteList(data)**

except *IOError* as ioerr:

print("IOError: " + ioerr)

return(None)

james = get\_data('james.txt')

julie = get\_data('julie.txt')

mikey = get\_data('mikey.txt')

sarah = get\_data('sarah.txt')

print("Athlete Name: " + james.name + ", DOB: " + james.dob + ", Top 3:" + *str*(james.best()))

## Web Development

### Pathnames

**import os**

**import glob**

os.path.dirname(\_\_file\_\_) 🡪 Directory of the current file

os.pardir 🡪 Go back to the previous folder

os.path.join(path1, path2) 🡪 Joins the path

os.path.abspath() 🡪 Absolute Path

data\_path = **os.path.abspath(**

**os.path.join(**

**os.path.dirname(\_\_file\_\_), os.pardir, “data/\*.txt”**

**)**

**)**

**glob.glob(data\_path)**

### Operating System

For Linux, Mac we need to prepare two things for our CGI script for execution

* Set the executables bit for our CGI using the chmod +x command
* Add **#! /usr/local/bin/python3**

### Show html json data through python

**import json**

**print('Content-type: text/html \n\n')**

print(json.dumps(get\_all\_data))

### To send form data

**import cgi**

form\_data = **cgi.FieldStorage()**

athlete\_name = **form\_data[‘which\_athlete’].value**

### httpd.py

**from http.server import HTTPServer, CGIHTTPRequestHandler**

port = **7800**

httpd = **HTTPServer(('', port), CGIHTTPRequestHandler)**

print("Starting simple\_http onport: " + *str*(httpd.server\_port))

**httpd.serve\_forever()**

### Folder Structure

* cgi-bin
  + athletelist.py
  + atheletemodel.py
  + generate\_list.py
  + generate\_timing\_data.py
* css
  + style.css
* data
  + james.txt
  + julie.txt
  + mikey.txt
  + sarah.txt
  + athlete.pickle
* images
  + coach-head.jpg
* js
  + main.js
* node\_modules 🡪 Many files
* template
  + show\_athlete.html
  + view\_athlete.html
* favicon.ico
* http.py
* index.html

### athletelist.py

*def* sanitize(*data*):

if '-' in data:

splitter = '-'

elif ':' in data:

splitter = ':'

else:

return data

(mins, secs) = data.split(splitter)

return (mins + "." + secs)

***class* AthleteList(*list*):**

*def* \_\_init\_\_(*self*, *athlete*):

self.name = athlete['name']

self.dob = athlete['dob']

self.times = athlete['times']

*def* best(*self*):

return(sorted(*set*([sanitize(each\_record) for each\_record in self.times]))[0:3])

### athletemodel.py

import pickle

import os

from athletelist import AthleteList

*def* get\_data(*filename*):

try:

with open(filename) as data:

data = data.readline()

data = data.strip().split(',')

data = {"name": data.pop(0), "dob": data.pop(0), "times": data}

return AthleteList(data)

except *IOError* as ioerr:

print("IOError: " + ioerr)

return(None)

*def* put\_to\_store(*filelist*):

all\_athletes = {}

for each\_file in filelist:

data = get\_data(each\_file)

all\_athletes[data.name] = data

try:

**data\_path = os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), os.pardir,** 'data/athlete.pickle'))

with open(data\_path, 'wb') as savefile:

pickle.dump(all\_athletes, savefile)

except *IOError* as ioerr:

print("Input/Output Error: " + *str*(ioerr))

*def* get\_from\_store():

all\_athletes = {}

try:

data\_path = os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), os.pardir, 'data/athlete.pickle'))

with open(data\_path, 'rb') as readfile:

all\_athletes = pickle.load(readfile)

except *IOError* as ioerr:

print("Input/Output Error: " + *str*(ioerr))

return all\_athletes

### generate\_list.py

**import glob**

**import os**

import athletemodel

**import json**

data\_path = **os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), os.pardir, 'data/\*.txt'))**

data\_files = **glob.glob(data\_path)**

athletemodel.put\_to\_store(data\_files)

athletes = athletemodel.get\_from\_store()

get\_all\_data = []

for each\_athlete in athletes:

get\_all\_data.append({"name": athletes[each\_athlete].name, "dob": athletes[each\_athlete].dob})

**print('Content-type: text/html \n\n')**

**print(json.dumps(get\_all\_data))**

### generate\_timing\_data.py

import glob

**import os**

import json

import athletemodel

from athletelist import sanitize

query\_string **= os.getenv('QUERY\_STRING')**

data\_split **=** query\_string.split('=')

dob\_data **=** data\_split[1]

data\_path = os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), os.pardir, 'data/\*.txt'))

data\_files = glob.glob(data\_path)

athletemodel.put\_to\_store(data\_files)

athletes = athletemodel.get\_from\_store()

athlete\_data = []

for each\_athlete in athletes:

if dob\_data == athletes[each\_athlete].dob:

athlete\_data.append({"name" : athletes[each\_athlete].name, "dob" : athletes[each\_athlete].dob,

"times" : sorted(*set*([sanitize(each\_record) for each\_record in athletes[each\_athlete].times]))[0:3]})

print('Content-type: text/html \n\n')

print(json.dumps(athlete\_data))

### @property

The use of the **@property** decorator allows the class’ method to appear like an attribute to user of the class. Therefore, **print(athlete.top3())** 🡪 **print(athlete.top3)**

## Data Delivery to your CGI script

import **cgi**

form = **cgi.FieldStorage()**

import **os**

addr = **os.environ[‘REMOTE\_ADDR’]**

host = **os.environ[‘REMOTE\_HOST’]**

method = **os.environ[‘REQUEST\_METHOD’]**

query\_string = **os.getenv('QUERY\_STRING')**

import **time**

current\_time = **time.section(time.localtime())**

import **sys**

print(… … … …, **file=sys.stderr**)

## Connecting to the Database

### Knowing the version

python

import sqlite3

sqlite3.version 🡪 2.6.0

sqlite3.sqlite\_version 🡪 3.21.0

### Create Table & Database

import **sqlite3**

connection = **sqlite3.connect**('../data/coachdata.sqlite')

**cursor = connection.cursor()**

**cursor.execute**(**"""**CREATE TABLE athletes(id INTEGER PRIMARY KEY **AUTOINCREMENT** **NOT NULL**, name TEXT NOT NULL, dob DATE NOT NULL)**"""**)

cursor.execute("""CREATE TABLE timing\_data(id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, athletes\_id INT NOT NULL, value TEXT NOT NULL)""")

**connection.commit()**

**connection.close()**

🡪 Importing sqlite3

🡪 Connection∷ connection = sqlite3.connect

🡪 Cursor∷ cursor = connection.cursor()

🡪 Execute∷ cursor.excecute(“”” STATEMENT OVER HERE “””)

🡪 COMMIT∷ connection.commit()

🡪 CLOSE∷ connection.close()

### Inserting the data

import sqlite3

connection = sqlite3.connect('../data/coachdata.sqlite')

cursor = connection.cursor()

name = "Shabuktagin Photon Khan"

dob = "Khan"

cursor.execute("INSERT INTO athletes(name, dob) **VALUES(?, ?)**", **(name, dob)**)

connection.commit()

connection.close()

### Selecting the data

cursor.execute**(“**SELECT id from athletes FROM name = ? and dob = ?”, (name, dob))

**cursor.fetchone()** 🡪 returns the next row of data

**cursor.fetchmany()** 🡪 returns multiple rows of data

**cursor.fetchall()** 🡪 returns all of the data

import sqlite3

connection = sqlite3.connect("../data/coachdata.sqlite")

cursor = connection.cursor()

cursor.execute("""SELECT \* FROM athletes""")

result = cursor.fetchall()

data = []

**for i in range(0, (len(result))):**

**data.append({"id": result[i][0], "name": result[i][1], "dob": result[i][2]})**

connection.commit()

connection.close()

print(data)

### Update the data

import sqlite3

connection = sqlite3.connect("../data/coachdata.sqlite")

cursor = connection.cursor()

name = 'Shabuktagin Photon Khan'

dob = '19991-08-23'

id = 1

cursor.execute("UPDATE athletes SET name = ?, dob = ? WHERE id = ?", (name, dob, id))

connection.commit()

connection.close()

### Delete the data

import sqlite3

connection = sqlite3.connect("../data/coachdata.sqlite")

cursor = connection.cursor()

id = '3'

cursor.execute("DELETE FROM athletes WHERE id = ?", id)

connection.commit()

connection.close()

# Django

### Django Basics

#### Where is my python installed?

First setup the global environment for python and it’s python scripts

* import os
* import sys
* os.path.dirname(sys.executable)

These following commands will let us know where python is installed.

#### Knowing the version of PIP, Django and Python

* pip –version
* pip install Django
* django –version

#### Installing the virtual environment

* pip install environ
* pip install virtualenvwrapper-win

In these way, we can have multiple version of pythons installed in our desktop without facing any issues

#### Creating the virtual environment

Choose the desired folder to be the virtual folder. In my case,

mkvirtualenv django 🡪 Created the virtual environment

rmvirtualenv django 🡪 Deletes the virtual environment

#### What to install in the virtual environment?

Install the packages that has been installed outside the environment

* pip install environ
* pip install Django
* pip install –user cookiecutter 🡪 {**Template Optional}**
* cookiecutter <https://github.com/pydanny/cookiecutter-django>
* pip install freeze 🡪 To see the dependencies **{Optional}**

Instead of using cookiecutter use django’s default startproject command

#### What to do in the environment?

pip freeze

pip freeze > requirements.txt

django-admin startproject mysite 🡪 **Creates a project**

Now go to the folder

*/django/mysite/*

#### Run the local Server

*/django/mysite/manage.py*

manage.py runserver 🡪 By Default to 8000

manage.py runserver 8080 🡪 To change the port

#### Import Types

|  |  |  |
| --- | --- | --- |
| **Code** | **Import** | **Usage** |
| from core.views import FoodMixins | absolute import | Use when importing from outside the current app |
| from .models import WaffleCone | explicit relative | Use when importing from another module in the current app |
| from models import WaffleCone | implicit relative | Often used when importing from another module in the current app, but not a good idea |

#### Things to keep in mind

* Apps are containers in the freezer
* Packages are containers still at the store, waiting to be installed as apps

#### Make an app

*/django/mysite/manage.py*

manage.py startapp polls

#### Go to the polls folder

*/django/mysite/polls/views.py*

Update the ***views.py*** with the following code

**from** **django.http** **import** HttpResponse

**def** index(request):

**return** HttpResponse("Hello, world. You're at the polls index.")

#### Create a urls.py file inside it

*/django/mysite/polls/urls.py*

from **django.urls** import **path**

from **. import views**

urlpatterns = [

**path('', views.index, *name*='index'),**

]

#### Update the url in mysite folder

*/django/mysite/mysite/urls.py*

from django.contrib import admin

from django.urls **import include, path**

urlpatterns = [

**path('polls/', include('polls.urls')),**

path('admin/', admin.site.urls),

]

## Database Setup

*/django/mysite/manage.py*

**python manage.py migrate**

The migrate command looks at the INSTALLED\_APPS setting and creates any necessary database tables according to the database settings in our mysite/settings.py file and the database migrations shipped with the app.

*/django/mysite/polls/models.py*

**import datetime**

**from django.db import models**

**from django.utils import timezone**

class Question(models.Model):

question\_text = models.CharField(max\_length=200)

pub\_date = models.DateTimeField('date published')

def \_\_str\_\_(self):

return self.question\_text

def was\_published\_recently(self):

return self.pub\_date >= timezone.now() - datetime.timedelta(days=1)

class Choice(models.Model):

question = models.ForeignKey(Question, on\_delete=models.CASCADE)

choice\_text = models.CharField(max\_length=200)

votes = models.IntegerField(default=0)

def \_\_str\_\_(self):

return self.choice\_text

To include the app in our project, we need to add a reference to its configuration class in the INSTALLED\_APPS setting. The **PollsConfig** class is in the *polls/apps.py file*, so its dotted path is **'polls.apps.PollsConfig'**. Edit the *mysite/settings.py* file and add that dotted path to the INSTALLED\_APPS setting.

*/django/mysite/settings.py*

INSTALLED\_APPS = [

'polls.apps.PollsConfig',

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

]

### Model change

To make the django know that we have made some changes in the model

**manage.py makemigrations polls**

### SQL Migrate

**manage.py sqlmigrate polls 0001**

### Invoke Python Shell

**python manage.py shell**