Contents

[Setup 2](#_Toc505344910)

[Url’s 2](#_Toc505344911)

[Frozen binaries 2](#_Toc505344912)

[General 2](#_Toc505344913)

[Indentation 2](#_Toc505344914)

[Multi line 2](#_Toc505344915)

[Quotes 3](#_Toc505344916)

[Multiple statements 3](#_Toc505344917)

[Variables 3](#_Toc505344918)

[Data types 3](#_Toc505344919)

[String 3](#_Toc505344920)

[Lists 3](#_Toc505344921)

[Tuple 3](#_Toc505344922)

[Dictionary 4](#_Toc505344923)

[Operators 4](#_Toc505344924)

[Loop 4](#_Toc505344925)

[del 4](#_Toc505344926)

[Functions 4](#_Toc505344927)

[Lambda 4](#_Toc505344928)

[Modules 5](#_Toc505344929)

[Executing code in a module 5](#_Toc505344930)

[Module search sequence 5](#_Toc505344931)

[Variable scoping 5](#_Toc505344932)

[dir() 5](#_Toc505344933)

[locals() and globals() 5](#_Toc505344934)

[reload() 6](#_Toc505344935)

[python package 6](#_Toc505344936)

[File operations 6](#_Toc505344937)

[Exceptions 6](#_Toc505344938)

# Setup

VSCode can be used to run python code. You can install Anaconda and install it. It has got all libraries you may need. To run notebook run ‘jupyter notebook’ at command prompt. This gives you to run python code in several grids. It’s very convenient. It runs in browser.

I used visual studio code for Python.

You can install any Python library with pip install, which is like npm install in node.js. Once you do all your installs then by using pip freeze command you can transfer all library names in requirements.txt file which will behave more like package.json file.

# VirtualEnv

You can install multiple virtual environments having different installations. You can switch between them.

pip install virtualenv

mkdir newproject

cd newproject

virtualenv env # creates a virtual environment as name ‘env’

env\scripts\activate # activates the virtual environment env

In above you have created a virtual environment ‘env’ in current folder. The command ‘env\scripts\activate’ activates the already created environment ‘env’. Now there will be a prompt as <env> and whatever you install will be installed in ‘env’. You can use pip install flask to install flask library in ‘env’. Then you can do ‘import flask’ in your python file if the ‘env’ is active.

# Cloudjiffy PostgreSQL database connecting with Python

* For connecting from browser you need to give host as “node15792-chisel.cloudjiffy.net” and port as 11035 as you defined while creating endpoint in Cloudjiffy
* For connecting from Cloudjiffy servers you need to give the IP address for host say “192.168.3.138” and 5432 for port

# Python packages and import from packages

It is advisable to break your Python program in various packages. A package in Python is basically a folder having a must \_\_init\_\_.py blank file and other .py code files. Suppose you have folder FlaskApp where you want to put in your application. Here are the steps:

* Create the main .py file say adam1-gql.py
* Create a virtual environment say ‘env’ as above and activate and pip install required libraries in it.
* Create a folder say myPackage. This folder will have some helper .py files. Say you have helper1.py, helper2.py and helper3.py files in the folder. Now create a blank \_\_init\_\_.py file in the folder. This folder will now act as python package myPackage
* You can now put the following line in the main python file adam1-gql.py

From myPackage import helper1, helper2, helper3. You can now use any variable, method or class in helper1, helper2 or helper3 as say helper1.myMethod(), helper2.myVariable and so on.

# Working with flask

<https://www.tutorialspoint.com/flask>

Create a virtual env by name ‘env’ as above and activate it. Now pip install flask. See the project hello.py in python/flask repository. This is basic flask application

Hello.py

from helloPackage import helloHelper

app = helloHelper.app

if \_\_name\_\_ == '\_\_main\_\_':

    app.run()

.

In folder helloPackage there is a file helloHelper.py

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route('/')

def hello\_world():

    return 'Hello, World!'

## GraphQL

I used virtual environment by name ‘env’

Pip install flask\_graphql graphene

If virtual environment env is active, then above libraries will be installed in env. For GraphQL you need to define Schema. In schema you have Query and resolvers. In query you define fields. For every field you need to define resolver which is a method of type resolve\_fieldName.

I successfully implemented GraphQL in Python which was used from browser using localhost:5000/graphql. Also I fully implemented GraphQL in Flask in cloudjiffy cloud. Entire stuff is explained in another document flask.

# JSON

Python has built-in package named as json. Import json. Python has data structures list, tuple and dictionary. All these three and other primitive types can be converted to json object / string and vise versa.

Json to python: json.loads() method

Python to Json: json.dumps() method

# Config

You can use json, yml or .py config files and keep them in .gitignore. The import json, import yaml will help you out in parsing json and yaml files.

# 31-07-2019

from \_\_future\_\_ import

Above statement is used to import objects from next version. In Python 2.7 you can include that line to be code compatible with Python 3

Python has multiple inheritance.

## Connecting sql anywhere from python

pip install sqlanydb

import sqlanydb

try:

conn = sqlanydb.connect(uid='dba', pwd='sql', eng='server', dbn='capi2019', host='kushserver')

except (Exception, sqlanydb.Error) as error :

print ("Error while connecting to Sql Anywhere", error)

Remember to use host=’kushserver’ if server is running in a different machine.

# Revisit 09-07-2019

Install: Downloaded msi and installed.

Good tutorial: <https://docs.python.org/3.5/tutorial/index.html>

## Notes

* Python does not use braces. Use indenting as blocks.
* No explicit variable declaration. Just assign variables and use.

Multiple assignments allowed. a = b = c = 1; a,b,c = 1,2,"john"

* Data types are Number, String, List, Tuple and Dictionary
* You can delete a variable as
  + del var1, var2, var3
* List[], tuple(), dictionary {}

# Url’s

//good tutorial

<https://www.tutorialspoint.com/python3/index.htm>

Database connections postgresql python

Pip install psycopg2

# Frozen binaries

Using third party tools python files can be converted to binary executable files for which python environment is not required. Py2exe is for windows convert to .exe files. These contain PVM (Python virtual machine).

# General

At interactive prompt blank line is considered as termination of multi-line statement.

## Indentation

Python does not use braces for denoting blocks of code. Line indentation is used for denoting block of code. Indentation is used in “if” statement. Indentation can be tab or one or more spaces.

This will throw error because there is no indentation:

If True:

print(‘true’)

Following is OK because there is indentation:

If True:

print(‘true’)

## Multi line

‘\’ is used for multi line

total = item\_one + \

item\_two + \

item\_three

## Quotes

Python uses single(‘), double (“) or triple(‘’’) or (“””) for string literals. Triple quotes is used for string over multiple lines.

## Multiple statements

; allows multi statements in single line.

import sys; x = 'foo'; sys.stdout.write(x + '\n')

## Variables

Python does not require explicit variable declaration. Multiple assignments of variables are allowed.

a = 1

b=2

c=3

a=b=c=d=10

print(a)

a, b, c = 1, 2, "john"

Here a=, b=2 and c=”john”

## Data types

Python has five data types. 1) Number, 2) String, 3) List, 4) Tuple, 5) Dictionary

### String

str = 'Hello World!'

print (str) # Prints complete string

print (str[0]) # Prints first character of the string

print (str[2:5]) # Prints characters starting from 3rd to 5th

print (str[2:]) # Prints string starting from 3rd character

print (str \* 2) # Prints string two times

print (str + "TEST") # Prints concatenated string

### Lists

Can have multiple data types. Enclosed in []

list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]

tinylist = [123, 'john']

print (list) # Prints complete list

print (list[0]) # Prints first element of the list

print (list[1:3]) # Prints elements starting from 2nd till 3rd

print (list[2:]) # Prints elements starting from 3rd element

print (tinylist \* 2) # Prints list two times

print (list + tinylist) # Prints concatenated lists

### Tuple

These are read only lists. Cannot be changed. Enclosed in ()

tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )

tinytuple = (123, 'john')

print (tuple) # Prints complete tuple

print (tuple[0]) # Prints first element of the tuple

print (tuple[1:3]) # Prints elements starting from 2nd till 3rd

print (tuple[2:]) # Prints elements starting from 3rd element

print (tinytuple \* 2) # Prints tuple two times

print (tuple + tinytuple) # Prints concatenated tuple

### Dictionary

Are key value pairs. Enclosed in {}. Values can be accessed and assigned by [].

dict = {}

dict['one'] = "This is one"

dict[2] = "This is two"

tinydict = {'name': 'john','code':6734, 'dept': 'sales'}

print (dict['one']) # Prints value for 'one' key

print (dict[2]) # Prints value for 2 key

print (tinydict) # Prints complete dictionary

print (tinydict.keys()) # Prints all the keys

print (tinydict.values()) # Prints all the values

Data type conversion functions are available. Such as int(x), str(x), tuple(s)

## Operators

==, !=, & Binary AND, | Binary OR, and, or, not, in, not in, is, is not

## Loop

for, while loops are available.

## del

You can use del myvar to delete the variable from memory.

## Functions

Code block of each function starts with : and is indented.

def fname(arg1,arg2):

* Functions begin with def keyword followed by function name and parameters in paranthes() followed by : and then indented code block
* First line of function is documentation string and is optional
* Last statement of a function is return statement with none or any value.
* Function arguments are positional and must be used in the same order.
* All the arguments passed to a function are by reference, so they change the original value.
* Python supports anonymous or lambda functions.

## Lambda

# Function definition is here

sum = lambda arg1, arg2: arg1 + arg2

# Now you can call sum as a function

print ("Value of total : ", sum( 10, 20 ))

print ("Value of total : ", sum( 20, 20 ))

## Modules

Logically organize code. Module is python object with several attributes which you can bind and reference. It is a python file consisting of classes, functions and variables. Module is loaded only once regardless of times it is imported. From … import allows you to import specific attributes of a module.

from modname import name1[, name2[, ... nameN]]

### Executing code in a module

This will print Fibonacci sequence

# Fibonacci numbers module

def fib(n): # return Fibonacci series up to n

result = []

a, b = 0, 1

while b < n:

result.append(b)

a, b = b, a + b

return result

if \_\_name\_\_ == "\_\_main\_\_":

f = fib(100)

print(f)

### Module search sequence

Current directory > PYTHONPATH (environment variable) > default path

### 

### Variable scoping

It is important to understand the variable scoping. There are global and local variables. Inside a function if you refer to a variable it assumes it to be local variable. In order to refer to global variable inside a function write global myvar.

### dir()

This function returns all names defined by a module.

import math

content = dir(math)

print (content)

Result

['\_\_doc\_\_', '\_\_file\_\_', '\_\_name\_\_', 'acos', 'asin', 'atan',

'atan2', 'ceil', 'cos', 'cosh', 'degrees', 'e', 'exp',

'fabs', 'floor', 'fmod', 'frexp', 'hypot', 'ldexp', 'log',

'log10', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh',

'sqrt', 'tan', 'tanh']

\_\_name\_\_ is module name, \_\_file\_\_ is file name.

### locals() and globals()

Give all the names in local and global namespace.

### reload()

reloads a module so that code inside it can be executed again.

reload(myModule)

### 

### python package

A python package can have several modules, sub packages and so on. One \_\_init\_\_.py can import all modules and make them available together.

### 

### File operations

Python has file object and os package. These two collectively takes care of all file input / output and directory operations.

### 

### Exceptions

There is matured exception handling with try … except … finally

# Classes / Objects

class Employee:

'Common base class for all employees'

empCount = 0

def \_\_init\_\_(self, name, salary):

self.name = name

self.salary = salary

Employee.empCount += 1

def displayCount(self):

print "Total Employee %d" % Employee.empCount

def displayEmployee(self):

print "Name : ", self.name, ", Salary: ", self.salary

* empCount is class variable which is shared by all instances. It can be assessed as Employee.empCount.
* \_\_init\_\_(self, name, salary) is constructor. The first argument self is provided by system automatically.
* There is also a \_\_del\_\_ mthod available which is analogous to destructor.

This would create first object of Employee class

emp1 = Employee("Zara", 2000)

This would create second object of Employee class

emp2 = Employee("Manni", 5000)

Following methods are also available for attributes

hasattr(emp1, 'salary') # Returns true if 'salary' attribute exists

getattr(emp1, 'salary') # Returns value of 'salary' attribute

setattr(emp1, 'salary', 7000) # Set attribute 'salary' at 7000

delattr(emp1, 'salary') # Delete attribute 'salary'

* Ideally you should create each class in separate file and then import the file.

## Inheritence

class SubClassName (ParentClass1[, ParentClass2, ...]):

'Optional class documentation string'

class\_suite

class Parent: # define parent class

parentAttr = 100

def \_\_init\_\_(self):

print ("Calling parent constructor")

def parentMethod(self):

print ('Calling parent method')

def setAttr(self, attr):

Parent.parentAttr = attr

def getAttr(self):

print ("Parent attribute :", Parent.parentAttr)

class Child(Parent): # define child class

def \_\_init\_\_(self):

print ("Calling child constructor")

def childMethod(self):

print ('Calling child method')

c = Child() # instance of child

c.childMethod() # child calls its method

c.parentMethod() # calls parent's method

c.setAttr(200) # again call parent's method

c.getAttr() # again call parent's method

Output

Calling child constructor

Calling child method

Calling parent method

Parent attribute : 200

Multiple inheritance is allowed

class C(A, B): # subclass of A and B

* invisible attribute of parent can be achieved by prefixing it with double underscores. Here \_\_secretCount is invisible to child class.

class JustCounter:

\_\_secretCount = 0

def count(self):

self.\_\_secretCount += 1

print (self.\_\_secretCount)

counter = JustCounter()

counter.count()

counter.count()

print (counter.\_\_secretCount) # will result in error

But still this secret attribute can be visible by following code

print (counter.\_JustCounter\_\_secretCount)

that is

object.className\_\_attrName

# Regular expression

Module re provides all such functionalities. Python provides pattern matching and search capabilities.

# CGI

Common gateway interface. It is a set of standards which defines how information be exchanged between web server and custom script.

Simple Python web server can be created using CGI.

# Database

Separate modules are available for each database vendor for Python.

Successfully created and ran send mail sending program in Python.

# Threads

Python can create several threads and do synchronize, communicate and manage them. It uses threading module which is pretty advanced.

# GUI

Various options for developing GUI are 1) TKinter, 2) wxPython, 3) PyQt, 4) JPython

I tested on TKinter and it works.