# Python

# pycharm

## formatting

Cntrl + alt + L

# Anaconda SSL error

conda config --set ssl\_verify false

# Linux basic commands

find ./GFG -name sample.txt

# Os

## Assure path exist

def assure\_path\_exists(path):

dir = os.path.dirname(path)

if not os.path.exists(dir):

os.makedirs(dir)

# convert python



## .py to .c

* Install cython

pip install cython or pip3 install cython

* Create file > test.py

print("Convert python to cython")

* In cmd

>cython test.py

* It will create a test.c file

## .py to .pyc

* Create file > test.py

print("Convert python to cython")

* In cmd
* > python
* >>> import py\_compile
* >>> py\_compile.compile(‘test.py’)
* This will create test.pyc file

## .py to .pyx

* Create Pythagoras.py

import time

def count\_triples(limit):

result = 0

for a in range(1, limit + 1):

for b in range(a + 1, limit + 1):

for c in range (b + 1, limit + 1):

if c \* c > a \* a + b \* b:

break

if c \* c == (a \* a + b \* b):

result += 1

return result

if \_\_name\_\_ == '\_\_main\_\_':

start = time.time()

result = count\_triples(1000)

duration = time.time() - start

print(result, duration)

* Python Pythagoras.py
  + 881 9.497968435287476
* Create pyth\_triples.pyx

def count\_triples(limit):

result = 0

for a in range(1, limit + 1):

for b in range(a + 1, limit + 1):

for c in range (b + 1, limit + 1):

if c \* c > a \* a + b \* b:

break

if c \* c == (a \* a + b \* b):

result += 1

return result

* In cmd
* pip install cython
* create main.py

import time

import pyximport; pyximport.install()

import pyth\_triples

def main():

start = time.time()

result = pyth\_triples.count\_triples(1000)

duration = time.time() - start

print(result, duration)

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

main()

* python main.py
  + 881 6.446093320846558
* Create file setup.py

from distutils.core import setup

from Cython.Build import cythonize

setup(

ext\_modules = cythonize("pyth\_triples.pyx")

)

* python setup.py build\_ext –inplace
* it will create pyth\_triples.c, pyth\_triples.cp36-win\_amd64.pyd and a build folder
* edit main.py

import time

# import pyximport; pyximport.install()

import pyth\_triples

def main():

start = time.time()

result = pyth\_triples.count\_triples(1000)

duration = time.time() - start

print(result, duration)

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

main()

* python main.py
  + 881 6.341957330703735
* Edit pyth\_triples.pyx as

### Cdef declarations

list

cdef list foo = []

string

cdef char\* c\_string = NULL

float

cdef float x = 5.0

file

cdef FILE\* p

cdef int x,y,z

cdef char \*s

cdef float x = 5.2 (single precision)

cdef double x = 40.5 (double precision)

cdef list languages

cdef dict abc\_dict

cdef object thing

def count\_triples(limit):

cdef int result = 0

cdef int a = 0

cdef int b = 0

cdef int c = 0

for a in range(1, limit + 1):

for b in range(a + 1, limit + 1):

for c in range (b + 1, limit + 1):

if c \* c > a \* a + b \* b:

break

if c \* c == (a \* a + b \* b):

result += 1

return result

* Delete or move file > pyth\_triples.c, pyth\_triples.cp36-win\_amd64.pyd
* Python main.py
  + 881 0.03200125694274902
* python setup.py build\_ext –i
* python main.py
* in ubuntu >
* python setup.py build\_ext -i
* It will crate an .so file in the same directory

## .py to exe

from tkinter import \*

root = Tk()

# root.iconbitmap('C:\page\Code-Sign\exe\treeleaf\_83777.ico')

topFrame = Frame(root)

topFrame.pack()

bottomFrame = Frame(root)

bottomFrame.pack(side=BOTTOM)

button1 = Button(topFrame, text='Button 1', fg="red")

button2 = Button(topFrame, text='Button 2', fg="blue")

button3 = Button(topFrame, text='Button 3', fg="green")

button4 = Button(bottomFrame, text='Button 4', fg="purple")

button1.pack(side=LEFT)

button2.pack(side=LEFT)

button3.pack(side=LEFT)

button4.pack(side=BOTTOM)

root.mainloop()



* pip install pyinstaller
* pyinstaller.exe --onefile -w --icon=treeleaf\_83777.ico myFile.py

## .py to exe – 2

> https://pypi.org/project/auto-py-to-exe/

# Debugger

* import pdb
* pdb.pm()

## Jupyter notebook

from IPython.core.debugger import set\_trace

set\_trace()

# pickling

## save

import pickle

# take user input to take the amount of data

number\_of\_data = int(input('Enter the number of data : '))

data = []

# take input of the data

for i in range(number\_of\_data):

raw = input('Enter data '+str(i)+' : ')

data.append(raw)

# open a file, where you ant to store the data

file = open('important', 'wb')

# or

file = open('index.pickle', 'wb')

# dump information to that file

pickle.dump(data, file)

# close the file

file.close()

## load

import pickle

# open a file, where you stored the pickled data

file = open('important', 'rb')

# dump information to that file

data = pickle.load(file)

# close the file

file.close()

print('Showing the pickled data:')

cnt = 0

for item in data:

print('The data ', cnt, ' is : ', item)

cnt += 1



import pdb

def transform(x, y):

pdb.set\_trace()

z = x+ y

z = 5

x = 50

y = 60

pdb.set\_trace()

transform(5, 10)

print('z = ' + str(z))

# Pandas

## Read csv

|  |
| --- |
| # importing Pandas library  import pandas as pd    pd.read\_csv(filepath\_or\_buffer = "pokemon.csv")    # makes the passed rows header  pd.read\_csv("pokemon.csv", header =[1, 2])    # make the passed column as index instead of 0, 1, 2, 3....  pd.read\_csv("pokemon.csv", index\_col ='Type')    # uses passed cols only for data frame  pd.read\_csv("pokemon.csv", usecols =["Type"])    # reutruns pandas series if there is only one colunmn  pd.read\_csv("pokemon.csv", usecols =["Type"],                                squeeze = True)    # skips the passed rows in new series  pd.read\_csv("pokemon.csv",              skiprows = [1, 2, 3, 4]) |

# Program to create series

import pandas as pd # Import Panda Library

# Program to Create series with scalar values

Data =[1, 3, 4, 5, 6, 2, 9] # Numeric data

# Creating series with default index values

s = pd.Series(Data)

# predefined index values

Index =['a', 'b', 'c', 'd', 'e', 'f', 'g']

# Creating series with predefined index values

si = pd.Series(Data, Index)

## When Data contains Dictionary

# Program to Create Dictionary series

dictionary ={'a':1, 'b':2, 'c':3, 'd':4, 'e':5}

# Creating series of Dictionary type

sd = pd.Series(dictionary)

## When Data contains Ndarray

# Program to Create ndarray series

Data =[[2, 3, 4], [5, 6, 7]] # Defining 2darray

# Creating series of 2darray

snd = pd.Series(Data)

## Creation of DataFrame

# Program to Create DataFrame

import pandas as pd # Import Library

a = pd.DataFrame(Data) # Create DataFrame with Data

## Datafame with headers

import pandas as pd

# Program to Create ndarray series

Data =[[2, 3, 4], [5, 6, 7]] # Defining 2darray

# Creating series of 2darray

snd = pd.Series(Data)

a = pd.DataFrame(Data)

a.to\_csv("test.csv", index=False,  header=["Letter", "Number", "Symbol"])

## One or more dictionaries

Here, Data can be:

* One or more dictionaries
* One or more Series
* 2D-numpy Ndarray

# Program to Create Data Frame with two dictionaries

dict1 ={'a':1, 'b':2, 'c':3, 'd':4}  # Define Dictionary 1

dict2 ={'a':5, 'b':6, 'c':7, 'd':8, 'e':9} # Define Dictionary 2

Data = {'first':dict1, 'second':dict2} # Define Data with dict1 and dict2

df = pd.DataFrame(Data) # Create DataFrame

|  |  |  |
| --- | --- | --- |
|  | first | second |
| a | 1 | 5 |
| b | 2 | 6 |
| c | 3 | 7 |
| d | 4 | 8 |
| e |  | 9 |

## When Data is Series

# Program to create Dataframe of three series

import pandas as pd

s1 = pd.Series([1, 3, 4, 5, 6, 2, 9])        # Define series 1

s2 = pd.Series([1.1, 3.5, 4.7, 5.8, 2.9, 9.3]) # Define series 2

s3 = pd.Series(['a', 'b', 'c', 'd', 'e'])    # Define series 3

Data ={'first':s1, 'second':s2, 'third':s3} # Define Data

dfseries = pd.DataFrame(Data)            # Create DataFrame

## When Data is 2D-numpy ndarray

Note: One constraint has to be maintained while creating DataFrame of 2D arrays – Dimensions of 2D array must be same.

# Program to create DataFrame from 2D array

import pandas as pd # Import Library

d1 =[[2, 3, 4], [5, 6, 7]] # Define 2d array 1

d2 =[[2, 4, 8], [1, 3, 9]] # Define 2d array 2

Data ={'first': d1, 'second': d2} # Define Data

df2d = pd.DataFrame(Data) # Create DataFrame

|  |  |  |
| --- | --- | --- |
|  | first | second |
| 0 | [2, 3, 4] | [2, 4, 8] |
| 1 | [5, 6, 7] | [1, 3, 9] |

## Save dataframe

from pandas import DataFrame

cars = {'Brand': ['Honda Civic','Toyota Corolla','Ford Focus','Audi A4'],

'Price': [22000,25000,27000,35000]

}

df = DataFrame(cars, columns= ['Brand', 'Price'])

export\_csv = df.to\_csv (r'C:\Users\Ron\Desktop\export\_dataframe.csv', index = None, header=True) #Don't forget to add '.csv' at the end of the path

print (df)

## dataframe with different row length – dict of list {“sample”: [1,2,3]}

df\_out = pd.DataFrame(dict([ (k,pd.Series(v)) for k,v in sents\_d.items() ]))

## Adding multiple sheet

import pandas as pd  
  
df1 = pd.DataFrame({'Data': ['a', 'b', 'c', 'd']})  
  
df2 = pd.DataFrame({'Data': [1, 2, 3, 4]})  
  
df3 = pd.DataFrame({'Data': [1.1, 1.2, 1.3, 1.4]})  
  
writer = pd.ExcelWriter('multiple.xlsx', engine='xlsxwriter')  
  
df1.to\_excel(writer, sheet\_name='Sheeta')  
  
df2.to\_excel(writer, sheet\_name='Sheetb')  
  
df3.to\_excel(writer, sheet\_name='Sheetc')  
  
writer.save()

# Threding

## Simple thread

import threading

import time

def sleeper(n, name):

print('Hi {} sleep 5 sec'.format(name))

time.sleep(n)

print('{} has woken up from sleep'.format(name))

t = threading.Thread(target=sleeper, name = 'thread1', args = (5,'thread1'))

t.start()

print('hello')

# after compleating exicution of thread the later code will exicute

t.join()

print('hello')

## threading cuncurrent

import threading

import time

def sleeper(n, name):

print('Hi {} sleep 5 sec'.format(name))

time.sleep(n)

print('{} has woken up from sleep'.format(name))

threads\_list = []

start = time.time()

for i in range(5):

t = threading.Thread(target = sleeper,

name = "thread{}".format(i),

args = (5, 'thread{}'.format(i)))

threads\_list.append(t)

t.start()

print('{} has started'.format(t.name))

for t in threads\_list:

t.join()

end = time.time()

print('time taken:', end-start)

print('all five threads finished')

normal code

start = time.time()

for i in range(5):

print("iteration %d started"%i)

sleeper(5,i)

end = time.time()

print('time taken:', end-start)

print('all five threads finished')

## Thread with return value

import concurrent.futures

def foo(bar):

print('hello {}'.format(bar))

return 'foo'

with concurrent.futures.ThreadPoolExecutor() as executor:

future = executor.submit(foo, 'world!')

return\_value = future.result()

print(return\_value)

## multithread with return value

import concurrent.futures

import time

URLS = ['http://www.foxnews.com/',

'http://www.cnn.com/',

'http://europe.wsj.com/',

'http://www.bbc.co.uk/',

'http://some-made-up-domain.com/']

# Retrieve a single page and report the URL and contents

def load\_url(url, timeout):

print("start ", url)

time.sleep(5)

return url, timeout

# We can use a with statement to ensure threads are cleaned up promptly

with concurrent.futures.ThreadPoolExecutor(max\_workers=5) as executor:

# Start the load operations and mark each future with its URL

future\_to\_url = {executor.submit(load\_url, url, 60): url for url in URLS}

for future in concurrent.futures.as\_completed(future\_to\_url):

url = future\_to\_url[future]

try:

data = future.result()

except Exception as exc:

print('%r generated an exception: %s' % (url, exc))

else:

print(data)

# database

## Sqlite3

### Create a Table

#!/usr/bin/python

import sqlite3

conn = sqlite3.connect('test.db')

print ("Opened database successfully")

conn.execute('''CREATE TABLE COMPANY

(ID INTEGER PRIMARY KEY AUTOINCREMENT,

NAME TEXT NOT NULL,

AGE INT NOT NULL,

ADDRESS CHAR(50),

SALARY REAL);''')

print ("Table created successfully")

conn.close()

# INTEGER PRIMARY KEY AUTOINCREMENT

### INSERT Operation

#!/usr/bin/python

import sqlite3

conn = sqlite3.connect('test.db')

print ("Opened database successfully")

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \

VALUES (1, 'Paul', 32, 'California', 20000.00 )");

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \

VALUES (2, 'Allen', 25, 'Texas', 15000.00 )");

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \

VALUES (3, 'Teddy', 23, 'Norway', 20000.00 )");

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \

VALUES (4, 'Mark', 25, 'Rich-Mond ', 65000.00 )");

conn.commit()

print ("Records created successfully")

conn.close()

### SELECT Operation

#!/usr/bin/python

import sqlite3

conn = sqlite3.connect('test.db')

print ("Opened database successfully")

cursor = conn.execute("SELECT id, name, address, salary from COMPANY")

for row in cursor:

print ("ID = ", row[0])

print ("NAME = ", row[1])

print ("ADDRESS = ", row[2])

print ("SALARY = ", row[3], "\n")

print ("Operation done successfully")

conn.close()

### UPDATE Operation

#!/usr/bin/python

import sqlite3

conn = sqlite3.connect('test.db')

print ("Opened database successfully")

conn.execute("UPDATE COMPANY set SALARY = 25000.00 where ID = 1")

conn.commit()

print ("Total number of rows updated :", conn.total\_changes)

cursor = conn.execute("SELECT id, name, address, salary from COMPANY")

for row in cursor:

print ("ID = ", row[0])

print ("NAME = ", row[1])

print ("ADDRESS = ", row[2])

print ("SALARY = ", row[3], "\n")

print ("Operation done successfully")

conn.close()

### DELETE Operation

#!/usr/bin/python

import sqlite3

conn = sqlite3.connect('test.db')

print ("Opened database successfully")

conn.execute("DELETE from COMPANY where ID = 2;")

conn.commit()

print ("Total number of rows deleted :", conn.total\_changes)

cursor = conn.execute("SELECT id, name, address, salary from COMPANY")

for row in cursor:

print ("ID = ", row[0])

print ("NAME = ", row[1])

print ("ADDRESS = ", row[2])

print ("SALARY = ", row[3], "\n")

print("Operation done successfully")

conn.close()

# Scheduler

* pip install schedule

import schedule

import time

def job():

print("I'm working...")

schedule.every(10).minutes.do(job)

schedule.every().hour.do(job)

schedule.every().day.at("10:30").do(job)

schedule.every(5).to(10).minutes.do(job)

schedule.every().monday.do(job)

schedule.every().wednesday.at("13:15").do(job)

schedule.every().minute.at(":17").do(job)

while True:

schedule.run\_pending()

time.sleep(1)

## Convert mp3 to wav

Download this zip [file](http://blog.gregzaal.com/how-to-install-ffmpeg-on-windows/#:~:text=If%20you%20try%20that%20right,and%20it'll%20understand%20us.)

Ffmpg [download](https://ffmpeg.zeranoe.com/builds/)

import subprocess

command = "ffmpeg -i temp.3gp -ab 160k -ac 2 -ar 44100 -vn audio.wav"

subprocess.call(command, shell=True)

## Voice call – twillo

<https://www.youtube.com/watch?v=-AChTCBoTUM>

import os

from twilio.rest import Client

account\_sid = "AC0ea4768da20941b347bcae28d99e5d77"

auth\_token = "d5e38b025fe43f6dc8a29f7a68c68538"

client =  Client(account\_sid, auth\_token)

call = client.calls.create(

    to = "+919447838962",

    from\_ = "+1 919 587 8152",

    # url = "https://demo.twilio.com/welcome/voice/"

    url = "https://demo.twilio.com/docs/voice.xml"

)

print(call.sid)

## .so file location

$ import sys

$ sys.path

* user lib python3 dist package –

## Telegram bot

<https://pypi.org/project/pyTelegramBotAPI/>

## Base64 encoding and decoding image

import base64

img2 = "jijo1.jpg"

with open(img1, "rb") as image\_file:

    encoded\_string = base64.b64encode(image\_file.read())

decode

import base64

from PIL import Image

from io import BytesIO

import PIL

face\_bs64 = request.POST['image1']

try:

    enc = base64.b64decode(face\_bs64)

    image1\_pil = PIL.Image.open(BytesIO(base64.b64decode(enc)))

    image1\_f = True

except:

    image1\_f = False

    status = 406

    error.append("Invalid image or base64 encryption error" )

# Shell script

## Face analytics start

#!/bin/sh

set -x

if [ -z "$STY" ]; then exec screen -dm -S face\_analytic /bin/bash "$0"; fi

cd /home/zcadmin/AI\_Projects/ImageProcessing\_DL

source  env/faceanalytics/bin/activate

cd faceanalytic

stunnel4 stunnel/dev\_https & python3 manage.py runserver 0.0.0.0:8444& HTTPS=1 python3 manage.py runserver 0.0.0.0:8000

## face analytics stop

#!/bin/sh

sudo netstat -tulpn | grep 8000 | awk {'print $7'} | cut -d "/" -f1 | xargs kill -9

sudo netstat -tulpn | grep 8444 | awk {'print $7'} | cut -d "/" -f1 | xargs kill -9

sudo netstat -tulpn | grep 8443 | awk {'print $7'} | cut -d "/" -f1 | xargs kill -9

ps -ef  |grep stunnel

# Sent mail

from email import encoders

from email.message import Message

from email.mime.base import MIMEBase

from email.mime.multipart import MIMEMultipart

from email.mime.text import MIMEText

import smtplib

signature = """

<br>

Warm regards, <br>

    <b style="color : red;">Switch Books</b> <br>

    +91 9447838962 | switchbooks.in

"""

fromaddr = "alancyriac111@gmail.com"

toaddr = "abhijithm2447@gmail.com"

msg = MIMEMultipart()

msg['From'] = fromaddr

msg['To'] = toaddr

msg['Subject'] = "Switch Books with sn"

body=("Hello Abhijith, Thank you for joining in the testing, regards by Switch Books" + signature)

msg.attach(MIMEText(body, 'html'))

server = smtplib.SMTP('smtp.gmail.com', 587)

server.starttls()

server.login(fromaddr, "Alancyriac@1996")

text = msg.as\_string()

server.sendmail(fromaddr, toaddr, text)

print("Mailed")

server.quit()

# E: Unable to locate package python3-pip

sudo add-apt-repository universe

sudo apt-get update

sudo apt-get install -y python3-pip

# Sent mail in linux cmd

sudo apt-get install ssmtp

# gksu gedit /etc/ssmtp/ssmtp.conf

Sudo vim /etc/ssmtp/ssmtp.conf

# The place where the mail goes. The actual machine name is required no

# MX records are consulted. Commonly mailhosts are named mail.domain.com

# mailhub=mail

mailhub = smtp.gmail.com:587

AuthUser = alancyriac111@gmail.com

AuthPass= Alancyriac@1996

UseTLS=YES

UseSTARTTLS=YES

# Where will the mail seem to come from?

#rewriteDomain=

rewriteDomain=gmail.com

# The full hostname

#hostname=ip-172-31-5-66.us-east-2.compute.internal

hostname=alancyriac111@gmail.com

# Are users allowed to set their own From: address?

# YES - Allow the user to specify their own From: address

# NO - Use the system generated From: address

FromLineOverride=YES

Vim save and exit

1. Switch to command mode by pressing the ESC key.
2. Press : (colon) to open the prompt bar in the bottom left corner of the window.
3. Type x after the colon and hit Enter. This will **save** the changes and **exit**.

Exit vim and type

$ ssmtp [alancyriac111@gmail.com](mailto:alancyriac111@gmail.com)

## method2

https://embedjournal.com/how-to-use-gmail-from-terminal-linux/

#Gmail account

defaults

#change the location of the log file to any desired location.

logfile ~/msmtp.log

account gmail

auth on

host smtp.gmail.com

from alancyriac111@gmail.com

auth on

tls on

tls\_trust\_file /etc/ssl/certs/ca-certificates.crt

user alancyriac111@gmail.com

password Alancyriac@1996

port 587

#set gmail as your default mail server.

account default : gmail

## sent mail office machine

import smtplib  
sender\_email = "manu.n@zerone-consulting.com"  
receiver\_email = "manu.n@zerone-consulting.com"  
message = """\  
Subject: Hi there  
This message is sent from Python new."""

smtp\_server = "192.168.0.5"  
port = 25 # For starttls  
sender\_email = "manu.n@zerone-consulting.com"  
password = ""  
# Try to log in to server and send email  
try:  
server = smtplib.SMTP(smtp\_server,port)

except Exception as e:  
print(e)

server.sendmail(sender\_email, receiver\_email, message)

## install heirloom-mailx

vim /etc/apt/sources.list

add below entry

deb http://security.ubuntu.com/ubuntu trusty-security main universe

save and close

sudo apt update

sudo apt install heirloom-mailx –y

# PDF creation

<https://pypi.org/project/pdfkit/>

from pyhtml2pdf import converter

converter.convert('https://pypi.org', 'sample.pdf')

# Jupyter change env

* ipython kernal --user --name=polyp

# Web scrapping

# Wget : not recognised

<https://www.youtube.com/watch?v=CkpTEJH6xkg>

download exe

<https://eternallybored.org/misc/wget/>

paste exe in C:\Windows\System32

# Profiler – runtime checker

<https://www.youtube.com/watch?v=qhb7cehwChc>

* python -m cProfile -o sample.prof demo.py
* snakeviz sample.prof

## Profiler for django

### Djanog-debug-toolbar

<https://www.youtube.com/watch?v=qWLk9S6mvAY>

* installatioin

<https://django-debug-toolbar.readthedocs.io/en/latest/installation.html#installation>

### dajngo cprofiler

<https://pypi.org/project/django-cprofile-middleware/>

# regular expression

https://www.youtube.com/watch?v=K8L6KVGG-7o

## match

import re

sentance = """abcdef

ABCDEF"""

pattern = re.compile(r'abc')

matches = pattern.finditer(sentance)

for match in matches:

    print(match)

<\_sre.SRE\_Match object; span=(0, 3), match='abc'>

## Escape char

sentance = """abcdef

ABCDEF

.

"""

pattern = re.compile(r'\.')

matches = pattern.finditer(sentance)

for match in matches:

    print(match)

<\_sre.SRE\_Match object; span=(15, 16), match='.'>

## . any char except new line

sentance = """abcdef

ABCDEF

.

"""

pattern = re.compile(r'.')

matches = pattern.finditer(sentance)

for match in matches:

    print(match)

<\_sre.SRE\_Match object; span=(0, 1), match='a'>

<\_sre.SRE\_Match object; span=(1, 2), match='b'>

<\_sre.SRE\_Match object; span=(2, 3), match='c'>

<\_sre.SRE\_Match object; span=(3, 4), match='d'>

<\_sre.SRE\_Match object; span=(4, 5), match='e'>

<\_sre.SRE\_Match object; span=(5, 6), match='f'>

<\_sre.SRE\_Match object; span=(6, 7), match=' '>

<\_sre.SRE\_Match object; span=(8, 9), match='A'>

<\_sre.SRE\_Match object; span=(9, 10), match='B'>

<\_sre.SRE\_Match object; span=(10, 11), match='C'>

<\_sre.SRE\_Match object; span=(11, 12), match='D'>

<\_sre.SRE\_Match object; span=(12, 13), match='E'>

<\_sre.SRE\_Match object; span=(13, 14), match='F'>

<\_sre.SRE\_Match object; span=(15, 16), match='.'>

## \d – digit between (0-9)

sentance = """abcdef

ABCDEF

.

11345

"""

pattern = re.compile(r'\d')

matches = pattern.finditer(sentance)

for match in matches:

    print(match)

<\_sre.SRE\_Match object; span=(17, 18), match='1'>

<\_sre.SRE\_Match object; span=(18, 19), match='1'>

<\_sre.SRE\_Match object; span=(19, 20), match='3'>

<\_sre.SRE\_Match object; span=(20, 21), match='4'>

<\_sre.SRE\_Match object; span=(21, 22), match='5'>

## \D – not a dig(0-9)

sentance = """abcdef

ABCDEF

.

11345

"""

pattern = re.compile(r'\D')

matches = pattern.finditer(sentance)

for match in matches:

    print(match)

<\_sre.SRE\_Match object; span=(0, 1), match='a'>

<\_sre.SRE\_Match object; span=(1, 2), match='b'>

<\_sre.SRE\_Match object; span=(2, 3), match='c'>

<\_sre.SRE\_Match object; span=(3, 4), match='d'>

<\_sre.SRE\_Match object; span=(4, 5), match='e'>

<\_sre.SRE\_Match object; span=(5, 6), match='f'>

<\_sre.SRE\_Match object; span=(6, 7), match=' '>

<\_sre.SRE\_Match object; span=(7, 8), match='\n'>

<\_sre.SRE\_Match object; span=(8, 9), match='A'>

<\_sre.SRE\_Match object; span=(9, 10), match='B'>

<\_sre.SRE\_Match object; span=(10, 11), match='C'>

<\_sre.SRE\_Match object; span=(11, 12), match='D'>

<\_sre.SRE\_Match object; span=(12, 13), match='E'>

<\_sre.SRE\_Match object; span=(13, 14), match='F'>

<\_sre.SRE\_Match object; span=(14, 15), match='\n'>

<\_sre.SRE\_Match object; span=(15, 16), match='.'>

<\_sre.SRE\_Match object; span=(16, 17), match='\n'>

<\_sre.SRE\_Match object; span=(22, 23), match='\n'>

## \w word char(a-z, A-Z, 0-9,\_)

## \W not word char(a-z, A-Z, 0-9,\_)

## \s whitespace (space, tab, newline)

## \S not whitespace (space, tab, newline)

## \b word boundary

sentance = """abcdefhi helloh

ABCDEF

.

11345

"""

pattern = re.compile(r'\bh')

matches = pattern.finditer(sentance)

for match in matches:

    print(match)

<\_sre.SRE\_Match object; span=(9, 10), match='h'>

## \B not word boundary

sentance = """abcdefhi helloh

ABCDEF

.

11345

"""

pattern = re.compile(r'\Bh')

matches = pattern.finditer(sentance)

for match in matches:

    print(match)

<\_sre.SRE\_Match object; span=(6, 7), match='h'>

<\_sre.SRE\_Match object; span=(14, 15), match='h'>

## ^ beginning of string

sentance = """abcdefhi helloh

ABCDEF

.

11345

"""

pattern = re.compile(r'^h')

matches = pattern.finditer(sentance)

for match in matches:

    print(match)

None

## $ end of string

## 555-666-7777 pattern

sentance = """555-666-7777

568.968.8766

65-99-35

"""

pattern = re.compile(r'\d\d\d.\d\d\d.\d\d\d\d')

matches = pattern.finditer(sentance)

for match in matches:

    print(match)

<\_sre.SRE\_Match object; span=(0, 12), match='555-666-7777'>

<\_sre.SRE\_Match object; span=(13, 25), match='568.968.8766'>

## [] pack

sentance = """555-666-7777

568.968.8766

568\*968\*8766

65-99-35

"""

pattern = re.compile(r'\d\d\d[-.]\d\d\d[-.]\d\d\d\d')

matches = pattern.finditer(sentance)

for match in matches:

    print(match)

<\_sre.SRE\_Match object; span=(0, 12), match='555-666-7777'>

<\_sre.SRE\_Match object; span=(13, 25), match='568.968.8766'>

sentance = """555-666-7777

568.968.8766

568\*968\*8766

600.968.8766

500-968-8766

"""

pattern = re.compile(r'[56]00[-.]\d\d\d[-.]\d\d\d\d')

matches = pattern.finditer(sentance)

for match in matches:

    print(match)

<\_sre.SRE\_Match object; span=(39, 51), match='600.968.8766'>

<\_sre.SRE\_Match object; span=(52, 64), match='500-968-8766'>

## [1-5] – range

sentance = """555-666-7727

568.968.8766

568\*968\*8766

600.968.8766

500-968-8766

"""

pattern = re.compile(r'[1-5]')

matches = pattern.finditer(sentance)

for match in matches:

    print(match)

<\_sre.SRE\_Match object; span=(0, 1), match='5'>

<\_sre.SRE\_Match object; span=(1, 2), match='5'>

<\_sre.SRE\_Match object; span=(2, 3), match='5'>

<\_sre.SRE\_Match object; span=(10, 11), match='2'>

<\_sre.SRE\_Match object; span=(13, 14), match='5'>

<\_sre.SRE\_Match object; span=(26, 27), match='5'>

<\_sre.SRE\_Match object; span=(52, 53), match='5'>

## [^a-zA-Z] not character

text = "this is w10221 \n test AA~"

pattern = re.compile(r'[^a-zA-Z]')

matches = pattern.finditer(text)

for m in matches:

    print(m)

<\_sre.SRE\_Match object; span=(4, 5), match=' '>

<\_sre.SRE\_Match object; span=(7, 8), match=' '>

<\_sre.SRE\_Match object; span=(9, 10), match='1'>

<\_sre.SRE\_Match object; span=(10, 11), match='0'>

<\_sre.SRE\_Match object; span=(11, 12), match='2'>

<\_sre.SRE\_Match object; span=(12, 13), match='2'>

<\_sre.SRE\_Match object; span=(13, 14), match='1'>

<\_sre.SRE\_Match object; span=(14, 15), match=' '>

<\_sre.SRE\_Match object; span=(15, 16), match='\n'>

<\_sre.SRE\_Match object; span=(16, 17), match=' '>

<\_sre.SRE\_Match object; span=(21, 22), match=' '>

<\_sre.SRE\_Match object; span=(24, 25), match='~'>

text = "cat mat bat"

pattern = re.compile(r'[^b]at')

matches = pattern.finditer(text)

for m in matches:

    print(m)

<\_sre.SRE\_Match object; span=(0, 3), match='cat'>

<\_sre.SRE\_Match object; span=(4, 7), match='mat'>

## ? zero or one

text = """Mr. Schafer

Mr Smit

Ms David

Mrs. Robinson

Mr. T"""

pattern = re.compile(r'Mr\.?')

matches = pattern.finditer(text)

for m in matches:

    print(m)

<\_sre.SRE\_Match object; span=(0, 3), match='Mr.'>

<\_sre.SRE\_Match object; span=(12, 14), match='Mr'>

<\_sre.SRE\_Match object; span=(29, 31), match='Mr'>

<\_sre.SRE\_Match object; span=(43, 46), match='Mr.'>

## Mrs/Ms Name matching

text = """Mr. Schafer

Mr Smit

Ms David

Mrs. Robinson

Mr. T"""

pattern = re.compile(r'M(r|s|rs)\.?\s[A-Z]\w\*')

matches = pattern.finditer(text)

for m in matches:

    print(m)

<\_sre.SRE\_Match object; span=(0, 11), match='Mr. Schafer'>

<\_sre.SRE\_Match object; span=(12, 19), match='Mr Smit'>

<\_sre.SRE\_Match object; span=(20, 28), match='Ms David'>

<\_sre.SRE\_Match object; span=(29, 42), match='Mrs. Robinson'>

<\_sre.SRE\_Match object; span=(43, 48), match='Mr. T'>

## Groupping ()

text = """

https://www.google.com

http://coreyms.com

https://youtube.com

https://www.nasa.gov

"""

pattern = re.compile(r'https?://(www\.)?(\w+)(\.\w+)')

matches = pattern.finditer(text)

for m in matches:

    print(m)

    print(m.group(2))

<\_sre.SRE\_Match object; span=(1, 23), match='https://www.google.com'>

google

<\_sre.SRE\_Match object; span=(24, 42), match='http://coreyms.com'>

coreyms

<\_sre.SRE\_Match object; span=(43, 62), match='https://youtube.com'>

youtube

<\_sre.SRE\_Match object; span=(63, 83), match='https://www.nasa.gov'>

Nasa

## Sub with group

text = """

https://www.google.com

http://coreyms.com

https://youtube.com

https://www.nasa.gov

"""

pattern = re.compile(r'https?://(www\.)?(\w+)(\.\w+)')

substituted = pattern.sub(r'\2\3', text)

print(substituted)

google.com

coreyms.com

youtube.com

nasa.gov

# Jupyter Notebook

<https://www.youtube.com/watch?v=wb6k_T4rKBQ&list=PLtPIclEQf-3fhfoFQU2MJYnQ6CyjQLQEa>

## Interactive widget

from IPython.display import display

import ipywidgets as widgets

import matplotlib.pyplot as plt

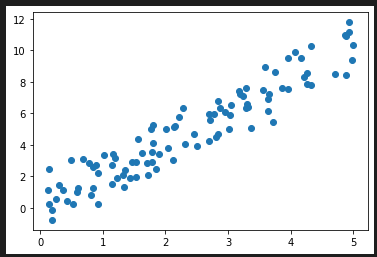
import numpy as np

x = np.random.uniform(0, 5, size=100)

ep = np.random.normal(size = 100)

y = 2\*x + ep

plt.scatter(x,y)



x\_values = np.linspace(0, 5 , 1000)

# slider

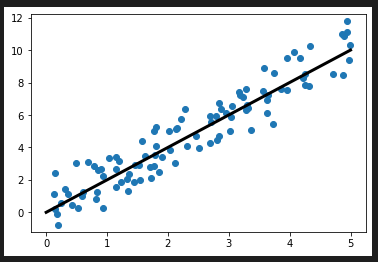
def slop\_viz(m=1):

    plt.scatter(x, y)

    plt.plot(x\_values, m \* x\_values, lw = 3, color="black")

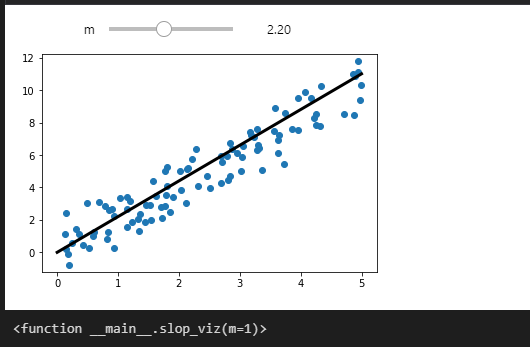
    plt.ylim(-1.2, 12.2)

slop\_viz(m=2)



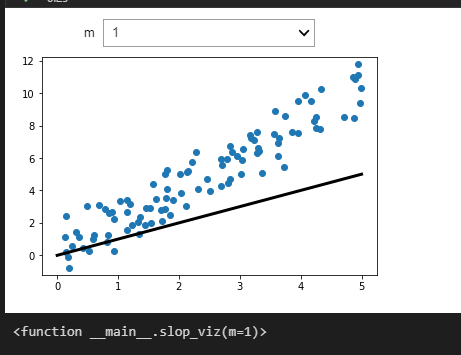
### # slider

widgets.interact(slop\_viz, m = (0.2, 5 , 0.2))



### # dropdown

widgets.interact(slop\_viz, m = [0,1,2,3,4,5])



### # checkbox

def slop\_viz\_updated(m=1, line = False, text = ""):

    plt.scatter(x, y)

    if line:

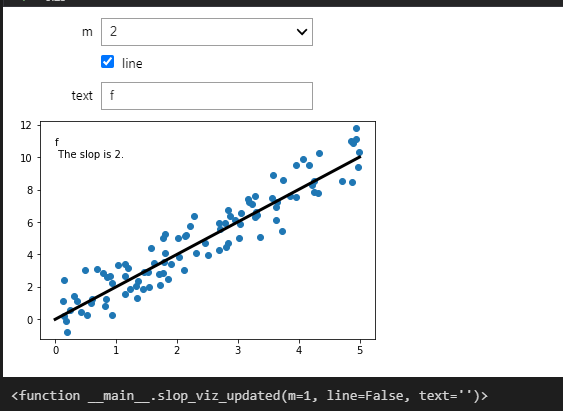
        plt.plot(x\_values, m \* x\_values, lw = 3, color="black")

    if text:

        plt.text(0, 10, f"{text}\n The slop is {m}.")

    plt.ylim(-1.2, 12.2)

widgets.interact(slop\_viz\_updated, m = [0,1,2,3,4,5], line = False, text="")



### # custom widget

radio\_btn = widgets.RadioButtons(

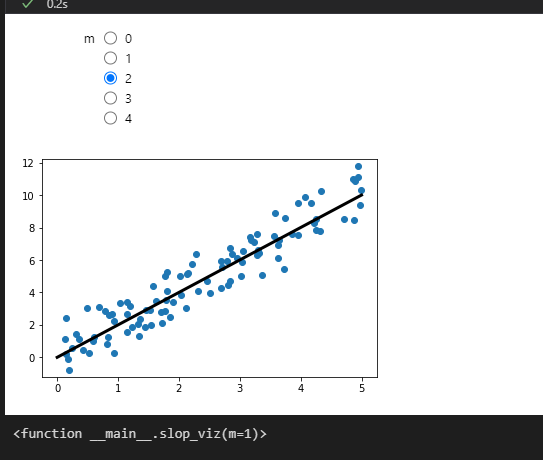
    options = [0,1,2,3,4],

    value = 1,

    desciption = "Slop:"

)

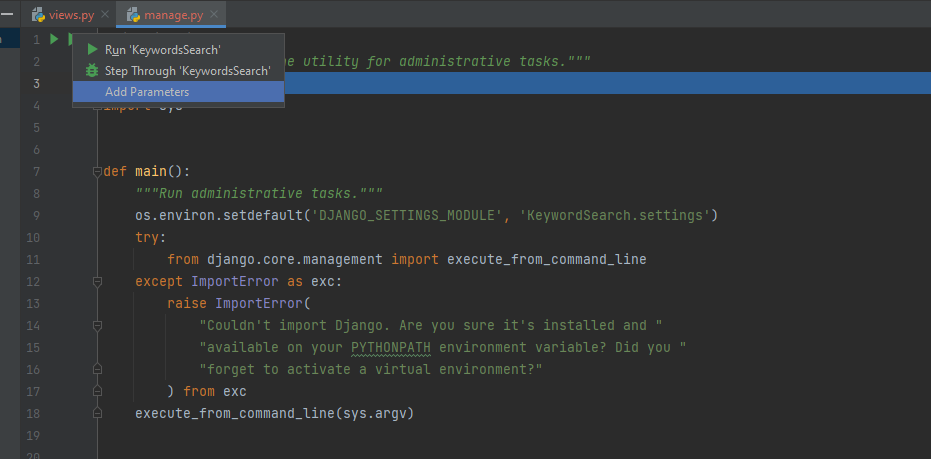
widgets.interact(slop\_viz, m= radio\_btn)

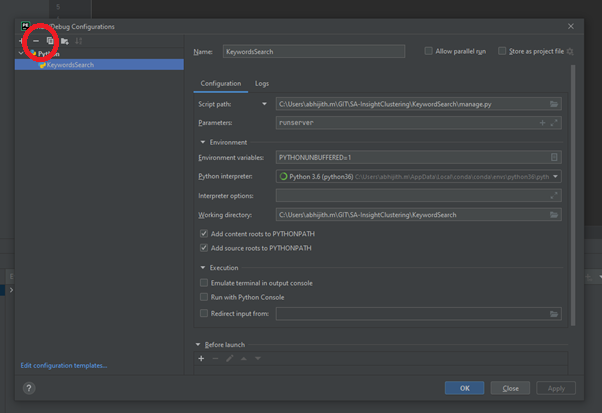


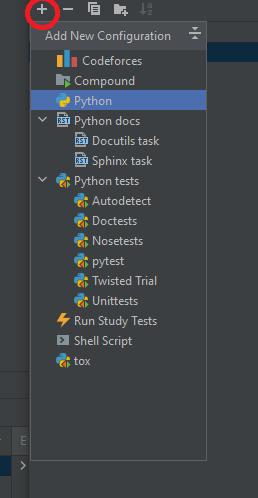
# Pycharm Django setup

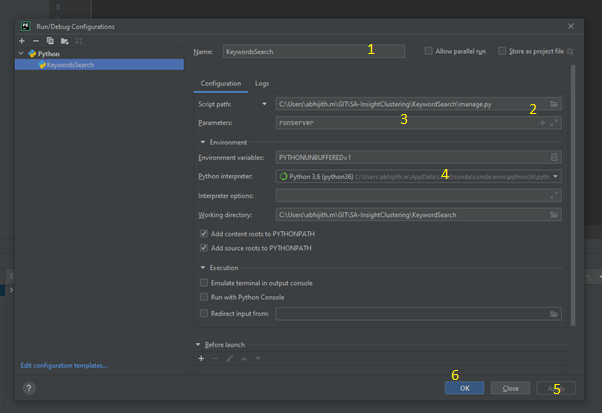
<https://www.youtube.com/watch?v=34Gum_6iGFM>

* right click run button









# Sort dictionary with value

<https://stackoverflow.com/a/613218/7360872>

# Sort list of dict

a = [{'name':'Homer', 'age':39}, {'name':'Homer', 'age':28}, {'name':'Homer', 'age':40}]  
sorted(a, key=lambda k : k['age'])

# Tricks and Tips

## Assert in Python

Assertions are meant to be internal self-checks for your program. assert statement is a debugging aid, not a mechanism for handling run-time errors. The goal of using assertions is to let developers find the likely root cause of a bug more quickly. assert statement into a null-operation: the assertions simply get compiled away and won’t be evaluated, which means that none of the conditional expressions will be executed.

def apply\_discount(product, discount):  
 price = int(product['price'] \* (1.0 - discount))  
 assert 0 <= price <= product['price'], "The price does not fall under 0"  
 return price  
  
  
shoes = {'name': 'Fancy Shoes', 'price': 14900}  
res = apply\_discount(shoes, 0.25)  
print(res)  
apply\_discount(shoes, 2.0)

## incorrect use of assert statements

what’s going to happen if assertions are disabled?

never use assertions to do data validation

def delete\_product(prod\_id, user):  
 assert user.is\_admin(), 'Must be admin'  
 assert store.has\_product(prod\_id), 'Unknown product'  
 store.get\_product(prod\_id).delete()

## Complacent Comma Placement

my\_str = ('This is a super long string constant '  
 'spread out across multiple lines. '  
 'And look, no backslash characters needed!')  
print(my\_str)

## Context Managers and the with Statement

with open('hello.txt', 'w') as f:  
 f.write('hello, world!')

What’s a context manager? It’s a simple “protocol” (or interface) that your object needs to follow in order to support the with statement. Basically, all you need to do is add \_\_enter\_\_ and \_\_exit\_\_ methods to an object if you want it to function as a context manager. Python will call these two methods at the appropriate times in the resource management cycle.

class ManagedFile:  
 def \_\_init\_\_(self, name):  
 self.name = name  
  
 def \_\_enter\_\_(self):  
 self.file = open(self.name, 'w')  
 return self.file  
  
 def \_\_exit\_\_(self, exc\_type, exc\_val, exc\_tb):  
 if self.file:  
 self.file.close()

with ManagedFile('hello.txt') as f:  
 f.write('hello, world!')  
 f.write('bye now')

contextlib.contextmanager decorator to define a generator-based factory function for a resource that will then automatically support the with statement.

from contextlib import contextmanager  
  
  
@contextmanager  
def managed\_file(name):  
 try:  
 f = open(name, 'w')  
 yield f  
 finally:  
 f.close()

with managed\_file('hello.txt') as f:  
 f.write('hello, world!')  
 f.write('bye now')

managed\_file() is a generator that first acquires the resource. The class-based implementation and the generator-based one are essentially equivalent.

class Indenter:  
 def \_\_init\_\_(self):  
 self.level = 0  
  
 def \_\_enter\_\_(self):  
 self.level += 1  
 return self  
  
 def \_\_exit\_\_(self, exc\_type, exc\_val, exc\_tb):  
 self.level -= 1  
  
 def print(self, text):  
 print(' ' \* self.level + text)  
  
  
with Indenter() as indent:  
 indent.print('hi!')  
 with indent:  
 indent.print('hello')  
 with indent:  
 indent.print('bonjour')  
 indent.print('hey')

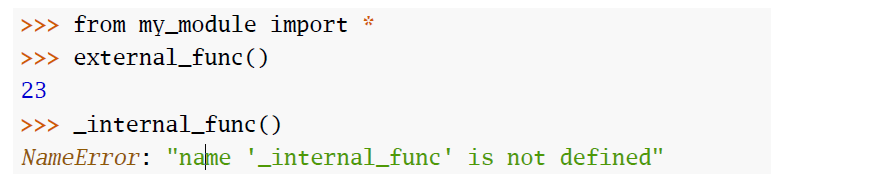
## Underscores, Dunders, and More

### Single Leading Underscore: “\_var”

The underscore prefix is meant as a hint to tell another programmer that a variable or method starting with a single underscore is intended for internal use. This convention is defined in PEP 8, the most commonly used Python code style guide.

Python does not have strong distinctions between “private” and “public” variables like Java does.

# my\_module.py:  
  
def external\_func():  
 return 23  
  
  
def \_internal\_func():  
 return 42



Unlike wildcard imports, regular imports are not affected by the leading single underscore naming convention:

Text

Description automatically generated with medium confidence

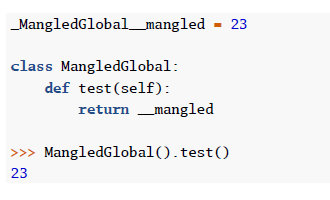
### Single Trailing Underscore: “var\_”

names like class or def cannot be used as variable names in Python. In this case, you can append a single underscore to break the naming conflict:

Text

Description automatically generated with medium confidence

### Double Leading Underscore: “\_\_var”



### Double Leading and Trailing Underscore:

Graphical user interface

Description automatically generated with medium confidence

### Single Underscore: “\_”

“\_” to indicate that it is just a temporary value:

A picture containing graphical user interface

Description automatically generated

This is handy if you’re working in an interpreter session and you’d like to access the result of a previous calculation:

Shape

Description automatically generated with medium confidence

**Key Takeaways**

• **Single Leading Underscore** “\_var”: Naming convention indicating

a name is meant for internal use. Generally not enforced

by the Python interpreter (except in wildcard imports)

and meant as a hint to the programmer only.

• **Single Trailing Underscore** “var\_”: Used by convention to

avoid naming conflicts with Python keywords.

• **Double Leading Underscore** “\_\_var”: Triggers name mangling

when used in a class context. Enforced by the Python interpreter.

• **Double Leading and Trailing Underscore** “\_\_var\_\_”: Indicates

special methods defined by the Python language. Avoid

this naming scheme for your own attributes.

• **Single Underscore** “\_”: Sometimes used as a name for temporary

or insignificant variables (“don’t care”). Also, it represents

the result of the last expression in a Python REPL session.

## String Formatting



### #1 – “Old Style” String Formatting

Text

Description automatically generated

### #2 – “New Style” String Formatting

name = "Bob"  
errno = 50159747054  
out = 'Hey {name}, there is a 0x{errno:x} error!'.format(name=name, errno=errno)  
print(out)

Hey Bob, there is a 0xbadc0ffee error!

### #3 – Literal String Interpolation (Python 3.6+)

name = "Bob"  
errno = 50159747054  
out = f"Hey {name}, there's a {errno:#x} error!"  
print(out)

Hey Bob, there's a 0xbadc0ffee error!

### #4 – Template Strings

from string import Template  
name = "Bob"  
errno = 50159747054  
templ\_string = 'Hey $name, there is a $error error!'  
out = Template(templ\_string).substitute(name=name, error=hex(errno))  
print(out)

Hey Bob, there is a 0xbadc0ffee error!

SECRET = 'this-is-a-secret'  
  
  
class Error:  
 def \_\_init\_\_(self):  
 pass  
  
  
err = Error()  
user\_input = '{error.\_\_init\_\_.\_\_globals\_\_[SECRET]}'  
print(user\_input.format(error=err))

this-is-a-secret

### Dan’s Python String Formatting Rule of Thumb:

*If your format strings are user-supplied, use Template*

*Strings to avoid security issues. Otherwise, use Literal*

*String Interpolation if you’re on Python 3.6+, and “New*

*Style” String Formatting if you’re not.*

## Effective Functions

def yell(text):  
 return text.upper() + '!'  
  
  
print(yell('hello'))  
del yell  
yell('hello?')

HELLO!

Traceback (most recent call last):

File "C:\AI\Projects\pythonProject\main.py", line 7, in <module>

yell('hello?')

NameError: name 'yell' is not defined

def yell(text):  
 return text.upper() + '!'  
  
bark = yell  
print(bark.\_\_name\_\_)

yell

### Functions Can Be Stored in Data Structures

def yell(text):  
 return text.upper() + '!'  
  
  
bark = yell  
funcs = [bark, str.lower, str.capitalize]  
for f in funcs:  
 print(f, f('hey there'))

<function yell at 0x0000023258B33E20> HEY THERE!

<method 'lower' of 'str' objects> hey there

<method 'capitalize' of 'str' objects> Hey there

### Functions Can Be Passed to Other Functions

def yell(text):  
 return text.upper() + '!'  
  
  
def greet(func):  
 greeting = func('Hi, I am a Python program')  
 print(greeting)  
  
bark = yell  
greet(bark)

HI, I AM A PYTHON PROGRAM!

def yell(text):  
 return text.upper() + '!'  
  
  
bark = yell  
print(list(map(bark, ['hello', 'hey', 'hi'])))

['HELLO!', 'HEY!', 'HI!']

### Functions Can Be Nested

def speak(text):  
 def whisper(t):  
 return t.lower() + '...'  
 return whisper(text)  
  
print(speak('Hello, World'))

hello, world...

def get\_speak\_func(volume):  
 def whisper(text):  
 return text.lower() + '...'  
  
 def yell(text):  
 return text.upper() + '!'  
  
 if volume > 0.5:  
 return yell  
 else:  
 return whisper  
  
  
print(get\_speak\_func(0.3))  
print(get\_speak\_func(0.7))  
speak\_func = get\_speak\_func(0.7)  
print(speak\_func('Hello'))

<function get\_speak\_func.<locals>.whisper at 0x0000018FCF59BF40>

<function get\_speak\_func.<locals>.yell at 0x0000018FCF59BEB0>

HELLO!

### Functions Can Capture Local State

def make\_adder(n):  
 def add(x):  
 return x + n  
  
 return add  
  
  
plus\_3 = make\_adder(3)  
plus\_5 = make\_adder(5)  
print(plus\_3(4))  
print(plus\_5(4))

7

9

### Objects Can Behave Like Functions

class Adder:  
 def \_\_init\_\_(self, n):  
 self.n = n  
  
 def \_\_call\_\_(self, x):  
 return self.n + x  
  
  
plus\_3 = Adder(3)  
print(plus\_3(4))

7

## Lambdas Are Single-Expression Functions

The lambda keyword in Python provides a shortcut for declaring small anonymous functions. They can be used whenever function objects are required.

print((lambda x, y: x + y)(5, 3))

8

I simply stated the expression I wanted to compute as part of a lambda, and then immediately evaluated it by calling the lambda expression like a regular function.

Technically, any time you’re expected to supply a function object you can use a lambda expression. And because lambdas can be anonymous, you don’t even need to assign them to a name first.

tuples = [(1, 'd'), (2, 'b'), (4, 'a'), (3, 'c')]  
print(sorted(tuples, key=lambda x: x[1]))

def make\_adder(n):  
 return lambda x: x + n  
  
  
plus\_3 = make\_adder(3)  
plus\_5 = make\_adder(5)  
print(plus\_3(4))  
print(plus\_5(4))

7

9

## Decorators

great use case for decoration

• logging

• enforcing access control and authentication

• instrumentation and timing functions

• rate-limiting

• caching, and more

def null\_decorator(func):  
 return func  
  
  
def greet():  
 return 'Hello!'  
  
  
greet = null\_decorator(greet)  
print(greet())

Hello!

def null\_decorator(func):  
 return func  
  
  
@null\_decorator  
def greet():  
 return 'Hello!'  
  
  
print(greet())

def uppercase(func):  
 def wrapper():  
 original\_result = func()  
 modified\_result = original\_result.upper()  
 return modified\_result  
  
 return wrapper  
  
  
@uppercase  
def greet():  
 return 'Hello!'  
  
  
print(greet())

HELLO!

def strong(func):  
 def wrapper():  
 return '<strong>' + func() + '</strong>'  
  
 return wrapper  
  
  
def emphasis(func):  
 def wrapper():  
 return '<em>' + func() + '</em>'  
  
 return wrapper  
  
  
@strong  
@emphasis  
def greet():  
 return 'Hello!'  
  
  
print(greet())

<em><strong>Hello!</strong></em>

def trace(func):  
 def wrapper(\*args, \*\*kwargs):  
 print(f'TRACE: calling {func.\_\_name\_\_}() '  
 f'with {args}, {kwargs}')  
 original\_result = func(\*args, \*\*kwargs)  
 print(f'TRACE: {func.\_\_name\_\_}() '  
 f'returned {original\_result!r}')  
 return original\_result  
  
 return wrapper  
  
  
@trace  
def say(name, line):  
 return f'{name}: {line}'  
  
  
print(say("hello world", "abhi"))

import functools  
  
  
def uppercase(func):  
 @functools.wraps(func)  
 def wrapper():  
 return func().upper()  
  
 return wrapper  
  
  
@uppercase  
def greet():  
 *"""Return a friendly greeting."""* return 'Hello!'  
  
  
print(greet.\_\_name\_\_)  
print(greet.\_\_doc\_\_)

## \*args and \*\*kwargs

def foo(required, \*args, \*\*kwargs):  
 print(required)  
 if args:  
 print(args)  
 if kwargs:  
 print(kwargs)  
  
  
foo('hello', 1, 2, 3, key1='value', key2=999)

hello

(1, 2, 3)

{'key1': 'value', 'key2': 999}

class Car:  
 def \_\_init\_\_(self, color, mileage):  
 self.color = color  
 self.mileage = mileage  
  
  
class AlwaysBlueCar(Car):  
 def \_\_init\_\_(self, \*args, \*\*kwargs):  
 super().\_\_init\_\_(\*args, \*\*kwargs)  
 self.color = 'blue'  
  
  
print(AlwaysBlueCar('green', 48392).color)  
print(AlwaysBlueCar('green', 48392).mileage)

blue

48392

def print\_vector(x, y, z):  
 print('<%s, %s, %s>' % (x, y, z))  
  
  
tuple\_vec = (1, 0, 1)  
print\_vector(tuple\_vec[0],  
 tuple\_vec[1],  
 tuple\_vec[2])  
dict\_vec = {'y': 0, 'z': 1, 'x': 1}  
print\_vector(\*tuple\_vec)  
print\_vector(\*\*dict\_vec)

<1, 0, 1>

<1, 0, 1>

<1, 0, 1>

Continuation in python.md file