Reference code blocks

```
## Reading from a file
f = open("readfile.txt", "r")

#Reads one line until '\n'
print f.readline()
print f.readline()

f.close()

# example
f= open ("readfile.txt", "r")
myList = []

for line in f:
    myList.append(line)
    print myList

f.close()

# example
a = open ("writefile.txt", "r")
print a.read()
a.close()
```

```
## Writing to a file
f = open ("writefile.txt", "w")
f.write("This is first line\n")
f.write("This is second line")
f.close()

# example
a = open ("writefile.txt", "r")
print a.read()
a.close()

# example: reading in buffer size and writing to file

def main():
    buffersize = 50000
    infile = open('bigfile.txt', 'r')
    outfile = open('new.txt', 'w')
    buffer = infile.read(buffersize)
    while len(buffer):
        outfile.write(buffer)
        buffer = infile.read(buffersize)

    print()
    print('Done')

    infile.close()
    outfile.close()

if __name__ == "__main__": main()
```

```
# read-write binary files

def main():
    infile = open('oliv.jpg', 'rb')
    outfile = open('new.jpg', 'wb')

    buffer = infile.read(buffersize)
    while len(buffer):
        outfile.write(buffer)
        #print('.', end = '' )
        buffer = infile.read(buffersize)

    print()
    print('Done')


if __name__ == "__main__": main()
```

```
# Example raising user defined exception

def main():
    try:
        for line in readfile('xlines.doc'):
print(line.strip())
    except IOError as e:
        print('cannot read file:', e)
    except ValueError as e:
        print('bad filename', e)

def readfile(filename):
    if filename.endswith('.txt'):
        fh = open(filename)
        return fh.readlines()
    else:
        raise ValueError('Filename must end with .txt')

             if __name__ == "__main__": main()
```

```
import sys

for line in sys.stdin:
    if line.strip() == 'exit':
        break
    print(f'Processing from stdin: ****** {line}*****')

O/P:
hello
Processing from stdin: ****** hello
*****
hello1
Processing from stdin: ****** hello1
*****
exit
```

```
EMPTY BLOCK
```

```
# file.tell() Tells current position bytes
fp = open("sample.txt", "r")
fp.read(8)

# Print the position of handle
print(fp.tell())

# Closing file
fp.close()

O/P: 8

# example
fp = open("sample2.txt", "wb")
print(fp.tell())

# Writing to file
fp.write(b'1010101')

print(fp.tell())

# Closing file
fp.close()

O/P:
0
7
```

```
# file.seek() moves the pointer to the position
# GfG.txt: "Code is like humor. When you have to explain it, it’s
bad."

f = open("GfG.txt", "r")

# Second parameter is by default 0
# sets Reference point to twentieth index position from the
# beginning f.seek(20)

# prints current position
print(f.tell())

print(f.readline())
f.close()

O/P: 20  and in next line,
When you have to explain it, it’s bad.

# example. (O/P: 47 and then in next line, its bad.
f.seek(-10, 2)

# prints current position
print(f.tell())

# Converting binary to string and printing

print(f.readline().decode('utf-8'))

f.close()
```

```
# reading a growing file

#!/usr/bin/python3

import time
import sys
import os

filename = sys.argv[1]

if not os.path.isfile(filename):
    print("invalid file name or file doesnot exist")

with open(filename) as fh:
    filesize = os.stat(filename)[6]
    fh.seek(filesize)   # move to end of file

    while True:
        where = fh.tell()
        line = fh.readline()
        if not line:
            time.sleep(1)
            fh.seek(where)
        else:
            print(line)
```

EMPTY BLOCK

```
# Exception
var1 = '1'
try:
        var1 = var1 + 1
except:
        print var1, " is not a
number"

print var1

# example
var1 = '1'
try:
        var2 = var1 + 10
except:
        var2 = int( var1 ) + 10

print "var2 is ", var2
```

```
## 1. Object is an instance of a class.
## 2. So classes makes objects and the functions in a class become the object's
methods.
## 3. Then which class function belongs to which instance of the class? So we just
implicitly
##    pass in objects property of self using def __init__ (self). Finally we get
down into the
##    heart of the initialize function using self.current to create an instance
variable.

## 1. Using classcalc.py
## 2. From classcalc tells python which file to import the classes from
## 3. import * means that we want to import all the classes from
## 4. myBuddy = Calculator() means we are initializing the object and that means
##    we can see the variable current of that myBuddy instance is set to zero.
```

```
# classcalc.py
class Calculator(object):
        # define class to simulate a
simple calculator
        def __init__(self):
                #start with zero
                self.current = 0
        def add(self, amount):
                #add number to current
                self.current += amount
        def getCurrent(self):
return self.current
```

```
classex1.py
############
from classcalc import *     # get classes from classcalc.py file
myBuddy = Calculator()   # creating a Calculator object named myBuddy
myBuddy.add(2)           # using myBuddy's add method
print myBuddy.getCurrent()    # calling getCurrent method of myBuddy to print
instance variable
```

```
# Generator: find primes upto 100.

## generator creates an iterator
def isprime(n):
    if n == 1:
        return False
    for x in range(2, n):
        if n % x == 0:
            return False
    else:
        return True

def primes( n = 1):
    while(True):
        if isprime(n): yield n
        n += 1

for n in primes():
    if n > 100: break
    print (n)
```

```
# find fibonacii numbers
## This __init__ function or method is called constructor.
## __init__ is called when the object is created using the f = Fibonacci
## Here __init__ is called with the reference to self and the variables a, b
## Constructor is optional. In this case we created to demonstrate.

class Fibonacci():
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def series(self):
        while(True):
            yield(self.b)
            self.a, self.b = self.b, self.a + self.b

f = Fibonacci(0, 1)
for r in f.series():
    if r > 100: break
    print r
```

```
# Threading

from threading import Thread
import time

def timer(name, delay, repeat):
        print "Timer: " + name + " Started"

        while repeat > 0 :
                time.sleep(delay)
                print name + ": " +
str(time.ctime(time.time()))
                repeat -= 1
        print "Timer: " + name + "Completed"

def main():
  t1 = Thread(target=timer, args=("Timer1", 1, 5))
```

```
# Threading: timer-lock.py

import threading
import time

## example below uses lock but you can also use semaphores
## semaphores allow more than one locks to be acquired
## If you have 10 threads that access your web page,
## you may allow max 2-3 threads to access your web page so that
they do not bring down the web server
## Use semaphore in that situations

tLock = threading.Lock()

def timer(name, delay, repeat):
        print "Timer: " + name + " Started"
```

```
  t2 = Thread(target=timer, args=("Timer2", 2, 5))

  t1.start()
  t2.start()
  print "Main Completed"

if __name__ == "__main__": main()
```

```
        tLock.acquire()
        print name + " has acquired the lock"
        while repeat > 0 :
                time.sleep(delay)
                print name + ": " + str(time.ctime(time.time()))
                repeat -= 1
        print name + " is releasing the lock"
        tLock.release()
        print "Timer: " + name + "Completed"

def main():
        t1 = threading.Thread(target=timer, args=("Timer1", 1, 5))
        t2 = threading.Thread(target=timer, args=("Timer2", 2, 5))

        t1.start()
        t2.start()
        print "Main Completed"

if __name__ == "__main__": main()
```

```
# database2.py    Database interaction

#!/usr/bin/python

## If it doesnot work then review the syntax in quick pythin book

import sqlite3

def insert(db, row):
    db.execue('insert into test(t1,i1) values (?,?)', (row['t1'],
row['i1']))
    db.commit()

def retrieve(db, t1):
    cursor = db.execute('select * from test where t1 = ?', (t1,))
    return cursor.fetchone()

def update(db, row):
    db.execute('update test set i1 = ? where t1 = ?', (row['i1'],
row['t1']))
    db.commit()

def delete(db, t1):
    db.execute('delete from test where t1 = ?', (t1,))
    db.commit()

def main():
    db = sqlite3.connect('test.db')
    db.row_factory = sqlite3.Row
    print('create table test')
    db.execute('drop table if exists test')
    db.execute('create table test(t1 text, i1 int)')

    print("Create rows")
    insert(db, dict(t1 = 'one', i1 = 1))
    insert(db, dict(t1 = 'two', i1 = 2))
    insert(db, dict(t1 = 'three', i1= 3))
    insert(db, dict(t1 = 'four', i1= 4))
    disp_rows(db)

    print("Retrieve rows")
    print( dict(retrieve(db,'one')), dict(retrieve(db, 'two')) )

    print("Update rows")
    update(db, dict(t1='one', i1 = 101))
    update(db, dict(t1='three', i1 = 103))
    disp_rows(db)

    print("Delete rows")
    delete(db,'one')
    delete(db, 'three')
    disp_rows(db)

if __name__ == "__main__": main()
```
