Handson #1 Give a Try PDH # 1- Welcome to Python Files :

File 2 :

```
fp = io.StringIO(zenPython)
    return fp
```

File 3 :

```
fp = io.StringIO(zenPython)
    zenlines=fp.readlines()[:5]
    return(zenlines)
```

File 4:

```
zenlines = [ line.strip() for line in zenlines ]
return zenlines
```

File 5:

```
    portions=re.findall(r"[-*] ?([^-*].*?) ?[-*]",zenPython)
```

==========

Handson #2 - Give a Try PDH # 2 -

```
 finalw = [re.sub(r'\bROAD\b', 'RD.', x) for x in addr]

 return finalw
```

==========

Handson #3 - Welcome to Python Database Connectivity

File 1 :

```
import sqlite3

def main():
    conn = sqlite3.connect('SAMPLE.db')
    #create connection cursor
    cursor = conn.cursor()
    #create table ITEMS using the cursor
    query = "CREATE TABLE ITEMS(item_id , item_name , item_descr , iption ,
item_category , quantity_in_stock)"

    cursor.execute(query)
    #commit connection
    conn.commit()
    #close connection
    conn.close()
```

File 2 :

```
def main():
    conn = sqlite3.connect('SAMPLE.db')
    cursor = conn.cursor()

    cursor.execute("drop table if exists ITEMS")

    sql_statement = '''CREATE TABLE ITEMS
    (item_id integer not null, item_name varchar(300),
    item_description text, item_category text,
    quantity_in_stock integer)'''
```

```
    cursor.execute(sql_statement)

    items = [(101, 'Nik D300', 'Nik D300', 'DSLR Camera', 3),
             (102, 'Can 1300', 'Can 1300', 'DSLR Camera', 5),
             (103, 'gPhone 13S', 'gPhone 13S', 'Mobile', 10),
             (104, 'Mic canvas', 'Mic canvas', 'Tab', 5),
             (105, 'SnDisk 10T', 'SnDisk 10T', 'Hard Drive', 1)
             ]

    #Add code to insert records to ITEM table

    sql = '''INSERT INTO ITEMS VALUES(?,?,?,?,?)'''

    try:

        cursor.executemany(sql,items)

        cursor.execute("select * from ITEMS")
    except:
        return 'Unable to perform the transaction.'
    rowout=[]
    for row in cursor.fetchall():
        rowout.append(row)
    return rowout
    conn.close()
```

File 3:

```
        cursor.execute("select * from ITEMS WHERE item_id < 103")
```

File 4:

```
cursor.executemany("update ITEMS set quantity_in_stock = ? where item_id = ?",
                       [(4, 103),
                        (2, 101),
                        (0, 105)])
```

File 5:

```
query1 = "delete from ITEMS where item_id = 105"
        cursor.execute(query1)
```

==========

Handson #4 : Higher Order Function and Closures1

File 1 - Closures

```
def detecter(element):
    def isIn(sequence):
        temp = 0
        for i in sequence:
            if i == element:
                temp = temp+1
        if temp > 0:
            return True
        else:
            return False

    return isIn

#Write closure function implementation for detect30 and detect45
```

```
detect30 = detecter(30)
detect45 = detecter(45)
```

File 2 :

```
def factory(n=0):

    def current():
        return n

    def counter():
        nonlocal n
        n += 1
        return n

    return current, counter

f_current,f_counter = factory(int(input()))
```

==========

Handson #5 : Welcome to Python - Decorators
[https://repl.it/@nimishmol/frescodecoratorfinaltest#main.py]

File 1:

```
def log(func):
    def inner(*args, **kwdargs):
        str_template = "Accessed the function -'{}' with arguments {}
".format(func.__name__,args)+"{}"
        return str_template
    return inner



@log
def greet(msg):
    return  msg
```

File 2:

```
@log
def average(n1,n2,n3):
    return (n1+n2+n3)/3
```

File 3:

```
def bold_tag(func):

    def inner(*args, **kwdargs):
        return '<b>'+func(*args, **kwdargs)+'</b>'

    return inner

@bold_tag
def say(msg):
    return msg
```

File 4:

```
#Implement italic_tag below
def italic_tag(func):
```

```
    def inner(*args, **kwdargs):
        return '<i>'+func(*args, **kwdargs)+'</i>'

    return inner

#Implement italic_tag below

@italic_tag
def say(msg):
    return msg

File 5:

@italic_tag
def greet():
    msg = 'Hello World! Welcome to Python Programming Language' #input()
    return msg

File 6:

@italic_tag
@bold_tag

#Add greet() implementation here
def greet():
    return input()

=========================================================

Handson # 6 : Welcome to Python - Give a Try - Defining an Abstract Class in
Python

class Animal(ABC):
    @abstractmethod
    def say(self):
        pass
# Define class Dog derived from Animal
# Also define 'say' method inside 'Dog' class
class Dog(Animal):
    def say(self):
        super().say()
        return("I speak Booooo")

=========================================================

Handson # 7 : Welcome to Python - Class and Static Methods

File 1 :

class Circle:
    no_of_circles = 0
    def __init__(self,radius):
        self.radius = radius
        Circle.no_of_circles += 1
    def area(self):
        return round((3.14*self.radius*self.radius),2)

File 2 :

class Circle:
    no_of_circles = 0
    def __init__(self,radius):
        self.radius = radius
        Circle.no_of_circles += 1
```

```
    def area(self):
        return round((3.14*self.radius*self.radius),2)
    @classmethod
    def getCircleCount(self):
        return Circle.no_of_circles
```

File 3:

```
class Circle(object):
    no_of_circles = 0
    def __init__(self,radius):
        self.radius = radius
        Circle.no_of_circles += 1
    @staticmethod
    def getPi():
        return 3.14
    def area(self):
        return round((self.getPi()*self.radius*self.radius),2)
    @classmethod
    def getCircleCount(self):
        return Circle.no_of_circles
```

==========

Handson # 8 Give a Try - Context Managers

File 1 :

```
with open(filename , 'w') as fp:
        content = fp.write(input_text)
```

File 2 :

```
def writeTo(filename, input_text):
    with open(filename , 'w') as fp:
        content = fp.write(input_text)
# Define the function 'archive' below, such that
# it archives 'filename' into the 'zipfile'
def archive(zfile, filename):
    with zipfile.ZipFile(zfile,'w') as zip:
        # writing each file one by one
            zip.write(filename)
```

File 3 :

```
 with subprocess.Popen(cmd_args, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
as p:
        out, err = p.communicate()
    return out
```

==========

Handson # 9 Give a Try - Coroutines

File 1 :

```
while True:
        n =yield
        t = (a*(n**2))+b
        string = "Expression, "+str(a)+"*x^2 + "+str(b)+", with x being "+str(n)
+" equals "+str(t)
        print(string)
```

File 2 :

```
def coroutine_decorator(coroutine_func):
    def wrapper(*args,**kwdargs):
        c = coroutine_func(*args,**kwdargs)
        next(c)
        return c
    return wrapper


# Define coroutine 'linear_equation' as specified in previous exercise
@coroutine_decorator
def linear_equation(a, b):
    while True:
        n =yield
        t = (a*(n**2))+b
        string = "Expression, "+str(a)+"*x^2 + "+str(b)+", with x being "+str(n)
+" equals "+str(t)
        print(string)
```

File 3:

```
def linear_equation(a, b):
    while True:
        n =yield
        t = (a*(n**2))+b
        string = "Expression, "+str(a)+"*x^2 + "+str(b)+", with x being "+str(n)
+" equals "+str(t)
        print(string)

# Define the coroutine function 'numberParser' below
def numberParser():
    equation1 = linear_equation(3, 4)
    equation2 = linear_equation(2, -1)
    # code to send the input number to both the linear equations
    next(equation1)
    equation1.send(6)
    next(equation2)
    equation2.send(6)

def main(x):
    n = numberParser()
    #n.send(x)
```

==========

Handson # 10 Descriptors

```
class Celsius:

    def __get__(self, instance, owner):
        return 5 * (instance.fahrenheit - 32) / 9

    def __set__(self, instance, value):
        instance.fahrenheit = 32 + 9 * value / 5

# Add temperature class implementation below.

class Temperature:

    celsius = Celsius()
```

```
def __init__(self, initial_f):
    self.fahrenheit = initial_f
```