While working on Fourier Series or some other Mathematical Problem, you might sometime have to work with Periodic Functions.

Periodic Functions are those that give the same value after a particular period.
So we will use this definition to define a periodic function in PYTHON.

Let's say that there is a function *f(x)* which is defined in the interval [li,lf] and is periodic with a period of *T=lf-li*.

Then the function should have the same value at: *f(x), f(x+T), f(x+2*T), ...*

i.e. *f(x)=f(x+T)=f(x+2*T)=*..... since period=T.

But I said that the function is defined only in the interval [li,lf]. So how is the computer supposed to calculate it's value at x>lf?
That's easy. Since the value of the function at *f(x+T)* is simply *f(x)*, therefore we can generalize that whenever *x>lf*: then,
f(x)=f(x-T). Note: We have to keep taking *x* back by *T i.e (x-T)* until it lies within [li,lf] where the function is well-defined.

Similarly what about the value of function at *x* less than *(li)* cause the function is not defined for values less than (li)?
Again, this time we use:*f(x)=f(x+T)*. Note: We keep translating *x* forward by *T i.e (x+T)* until it lies within [li,lf] where the function is well-defined.

Using the above two arguments we can create a function which will make any given function defined within [li,lf] and with a period T a periodic function.

Here is the code for that:

```
# Function that will convert any given function 'f' defined in a given range '[li,lf]'
to a periodic function of period 'lf-li'
def periodicf(li,lf,f,x):
    if x>=li and x<=lf :
        return f(x)
    elif x>lf:
        x_new=x-(lf-li)
        return periodicf(li,lf,f,x_new)
    elif x<(li):
        x_new=x+(lf-li)
        return periodicf(li,lf,f,x_new)
```

In the above code I have created a function 'periodicf' which takes as arguments the limits [li,lf] for which the function 'f' (third argument) is originally defined and the fourth argument is the value of x at which I want the value of 'f'.

The above code assumes the function to be defined within [li,lf] therefore the function's starting point is (li). However, if you want to create a function that is defined in a different manner then you will have to define 'f' correspondingly.

Following is a program which you can use to define various periodic functions such as sawtooth wave, cycloids, square wave, and triangular wave.

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.pyplot import *
```

```
fig = figure(figsize=(8, 8), dpi=120)



# Function that will convert any given function 'f' defined in a given range '[li,lf]'
to a periodic function of period 'lf-li'
def periodicf(li,lf,f,x):
    if x>=li and x<=lf :
        return f(x)
    elif x>lf:
        x_new=x-(lf-li)
        return periodicf(li,lf,f,x_new)
    elif x<(li):
        x_new=x+(lf-li)
        return periodicf(li,lf,f,x_new)

# The periodic version of sawtooth function
def sawtoothP(li,lf,x):
    return periodicf(li,lf,sawtooth,x)

# Non-periodic sawtooth function defined for a range [-l,l]
def sawtooth(x):
    return x

# The periodic version of square function
def squareP(li,lf,x):
    return periodicf(li,lf,square,x)

# Non-periodic square wave function defined for a range [-l,l]
def square(x):
    if x>0:
        return 5
    else:
        return 0

# The periodic version of triangle function
def triangleP(li,lf,x):
    return periodicf(li,lf,triangle,x)

# Non-periodic triangle wave function defined for a range [-l,l]
def triangle(x):
    if x>0:
        return x
    else:
        return -x

# The periodic version of cycloid function
def cycloidP(li,lf,x):
    return periodicf(li,lf,cycloid,x)

# Non-periodic cycloid wave function defined for a range [-l,l]
def cycloid(x):
```

```
    return np.sqrt(5**2-x**2)



if __name__ == "__main__":


    # plt.style.use('dark_background')
    plt.style.use('seaborn')
    plt.title('Periodic functions\nCycloid')


    li = -5
    lf = 5
    step_size = 0.05

    x_l = -20
    x_u = 50

    x = np.arange(x_l,x_u,step_size)
    y1 = [sawtoothP(li,lf,xi) for xi in x]
    y2 = [squareP(li,lf,xi) for xi in x]
    y3 = [triangleP(li,lf,xi) for xi in x]
    y4 = [cycloidP(li,lf,xi) for xi in x]

    x_plot =[]
    y_plot1 = []
    y_plot2 = []
    y_plot3 = []
    y_plot4 = []

    x_l_plot = x_l - 15
    x_u_plot = x_l_plot + 20
    plt.xlim(x_l_plot,x_u_plot)
    plt.ylim(-6,7)

    for i in range(x.size):
        x_plot.append(x[i])
        y_plot1.append(y1[i])
        y_plot2.append(y2[i])
        y_plot3.append(y3[i])
        y_plot4.append(y4[i])
        #Sawtooth
        plt.plot(x_plot,y_plot1,c='darkkhaki')
        #Square
        plt.plot(x_plot,y_plot2,c='tomato')
        #Triangular
        plt.plot(x_plot,y_plot3,c='orange')
        #Cycloid
        plt.plot(x_plot,y_plot4,c='slateblue')
        x_l_plot = x_l_plot + step_size
        x_u_plot = x_u_plot + step_size
```

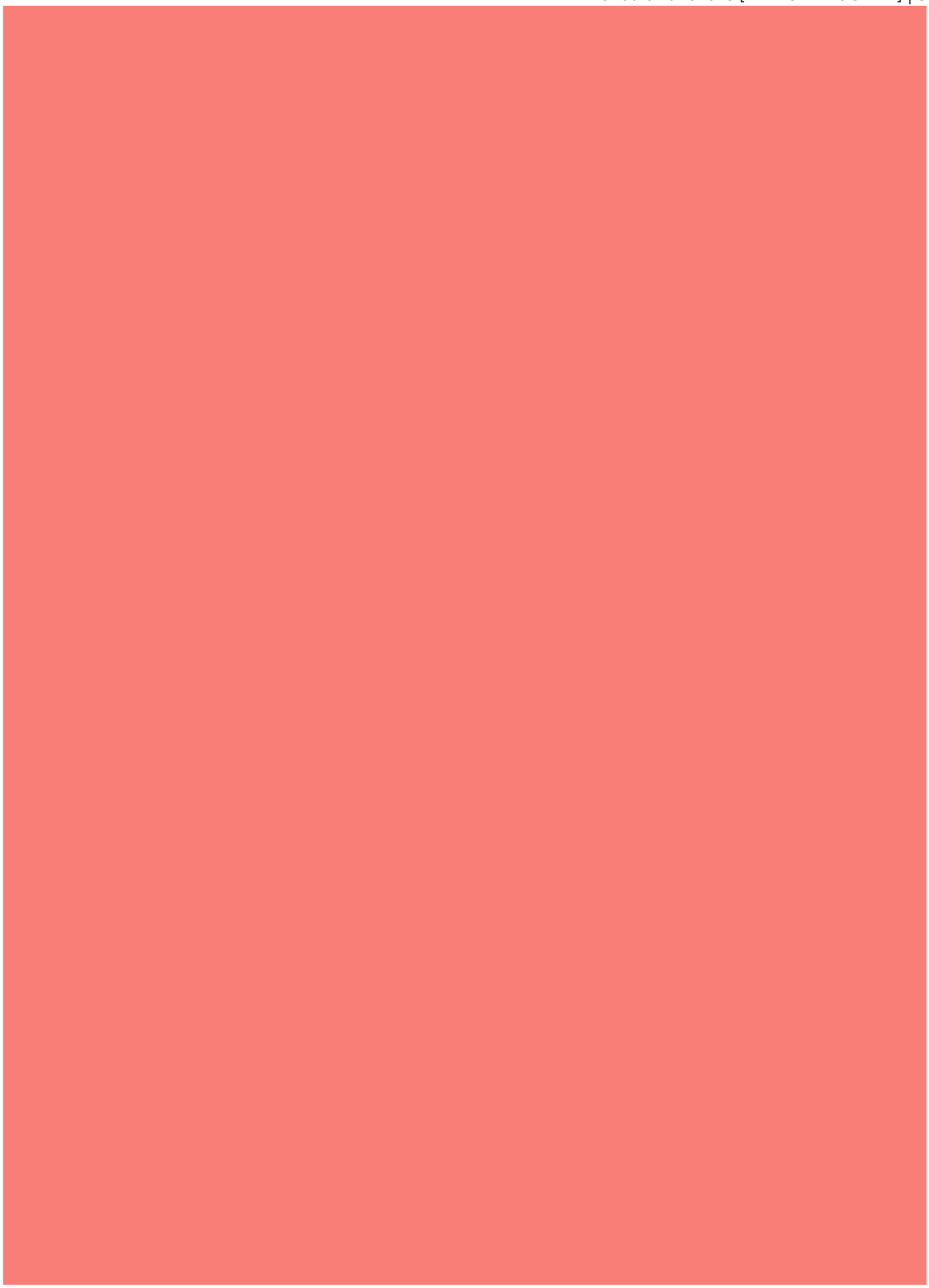
```
        plt.xlim(x_l_plot,x_u_plot)
        plt.pause(0.1)
    plt.show()
```

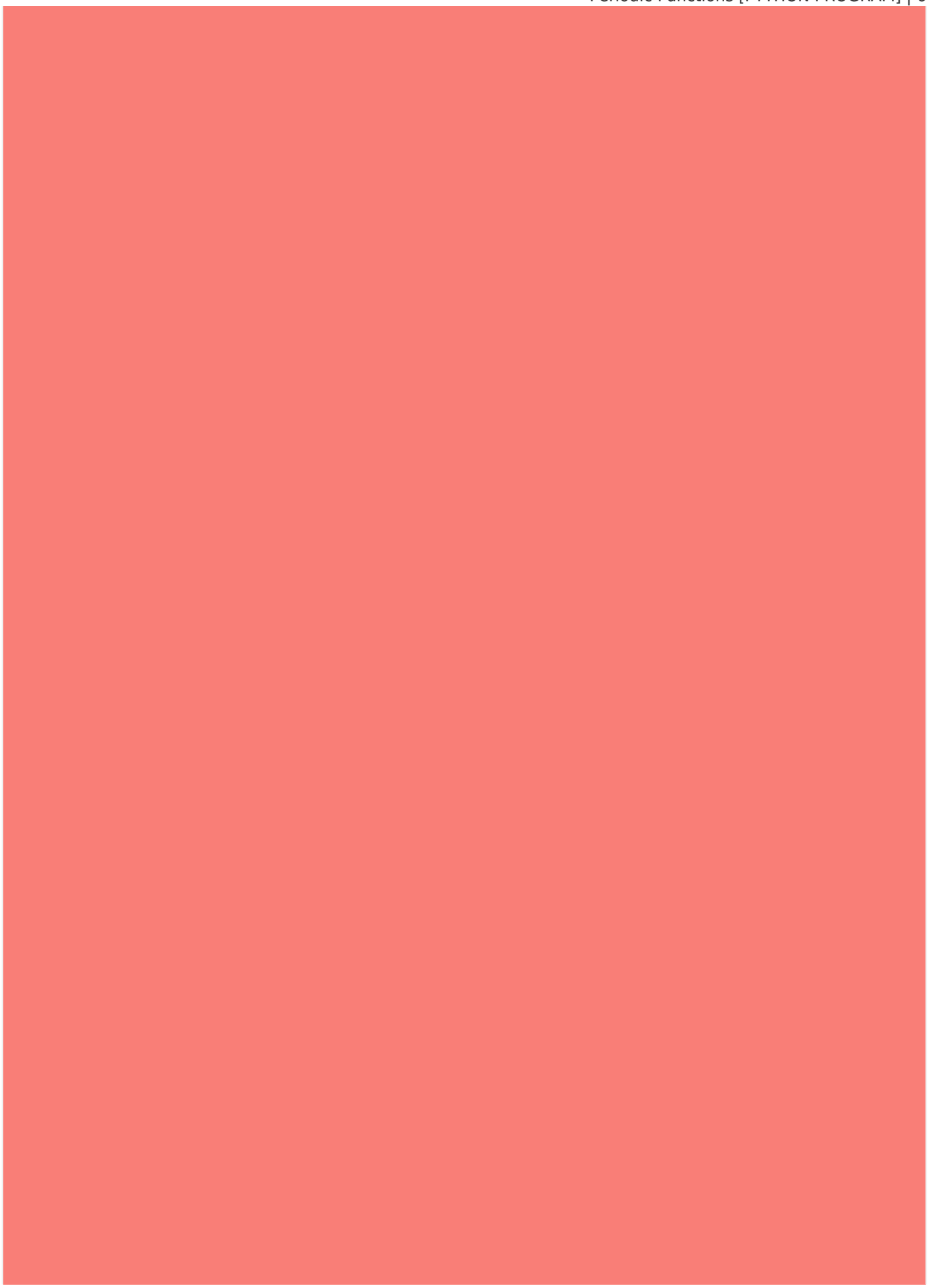
OUTPUT

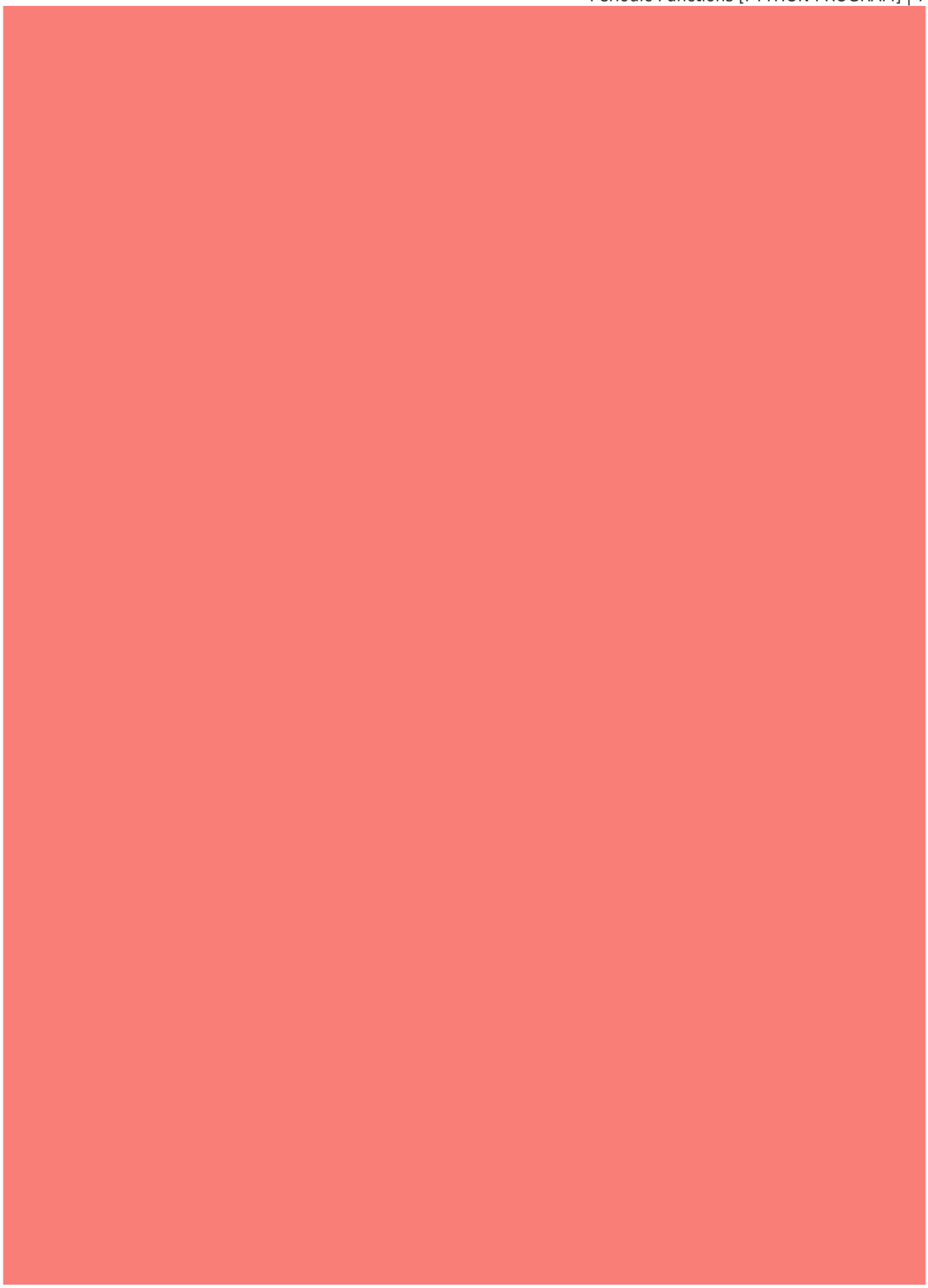
Manas Sharma

I'm a physicist specializing in computational material science with a PhD in Physics from Friedrich-Schiller University Jena, Germany. I write efficient codes for simulating light-matter interactions at atomic scales. I like to develop Physics, DFT, and Machine Learning related apps and software from time to time. Can code in most of the popular languages. I like to share my knowledge in Physics and applications using this Blog and a YouTube channel.

manas.bragitoff.com/







Share this:

Click to share on Facebook (Opens in new window)

Click to share on Twitter (Opens in new window)

Click to share on WhatsApp (Opens in new window)

Click to share on Pinterest (Opens in new window)

Click to share on Reddit (Opens in new window)

Click to share on LinkedIn (Opens in new window)

Click to email a link to a friend (Opens in new window)

Click to print (Opens in new window)

Click to share on Tumblr (Opens in new window)

Click to share on Pocket (Opens in new window)

Click to share on Telegram (Opens in new window)

[wpedon id="7041" align="center"]