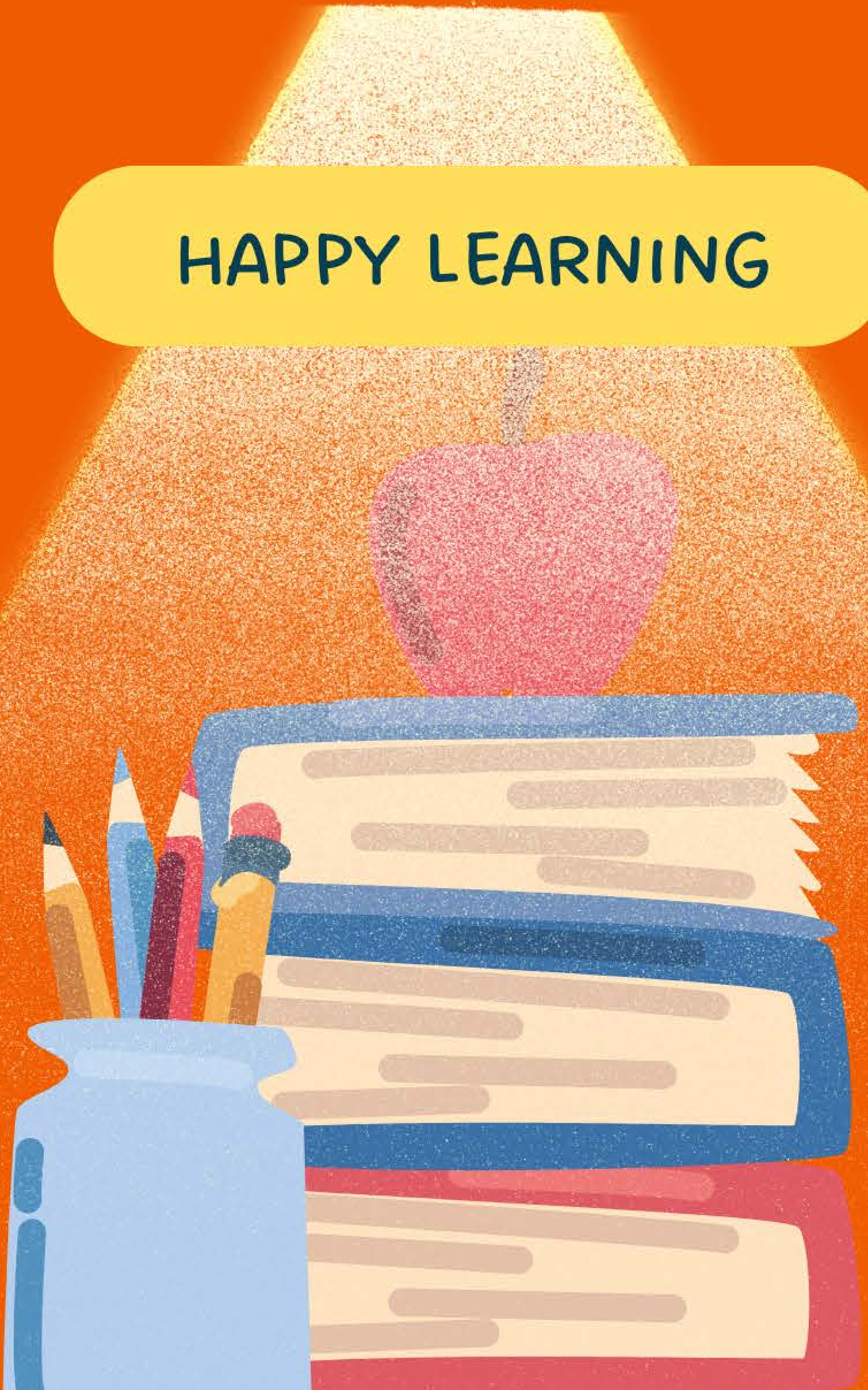
DIGYQUANT
ANALYTICS

PYTHON

FUNCTIONS & LAMBDA FUNCTIONS



www.digyquant.com

Functions || Try Except Block || Lambda Functions

In [1]:

```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt


img = Image.open('img.png')
data = np.asarray(img)
fig = plt.figure(figsize=(4,2))
ax = plt.Axes(fig, [0., 0., 1., 1.], )
ax.set_axis_off()
fig.add_axes(ax)
ax.imshow(data, aspect='auto')
plt.show()
```

Functions are of two types in Python- Built-in and User defined.

Functions are set of lines of code that are directed to perform a specific task.

Functions runs once a call is made to it. We can also pass arguments in function.

There are two types of arguments- keyword argument and positional arguments.

Functions are defined using def keyword.

Write a function to add two numbers

In [15]:

```
def twosum(x,y):
    z=x+y
    print("The sum is :",z)

twosum(20,30)
```

```
The sum is : 50
```

In [16]:

```
def twosum(x,y): # positional arguments
    z=x+y
    return z

a=twosum(20,30)
print("The sum is :",a)
```

```
The sum is : 50
```

In [17]:

```
def twosum(x=10,y=20): # keyword arguments
    z=x+y
    return z

a=twosum()
print("The sum is :",a)
```

```
The sum is : 30
```

In [18]:

```
def twosum(x=10,y=20): # keyword arguments
    z=x+y
    return z

a=twosum(40,50)
print("The sum is :",a)
```

```
The sum is : 90
```

Try Except Else block in python

The try block checks the code for any errors

The except block handles the error

In else block the code is executed if there is no error

Common types of built-in error types:

1.ValueError is raised if invalid value is provided.

2.TypeError is raised when datatypes are not approprate.

3.ZeroDivisionErroris raised when division by zero encountered.

4.SyntaxError is raised when there is an error in code.

In [1]:

```
try:
    "str"/20
except TypeError:
    print("Datatype is not appropriate")
```

```
Datatype is not appropriate
```

In [2]:

```
try:
    x=20/0
except ZeroDivisionError:
    print("Division by Zero is encountered")
```

```
Division by Zero is encountered
```

In [ ]:

```
try:
    x=int(input("Enter a number "))
    print(x)
except ValueError:
    print("Invalid Input")
```

Write a python function to find factorial of a number

!5= 5x4x3x2x1=120

In [ ]:

```
def fact_func():
    fact=1
    try:
        x=int(input("Enter a Number "))
        assert x>0 # it will return a boolean value
        if x==0:
            print("The factorial of 0 is 1")
    except ValueError:
        print("Enter a Numeric value")
    except AssertionError:
        print("Number cannot be negative")
    else:
        for i in range(1,x+1):
            fact=fact*i
        print("The factorial of {} is {}".format(x,fact))

fact_func()
```

In [3]:

```
x=5
fact=1
if x<0:
    print("There exists no factors for negative numbers")
elif x==0:
    print("Factorial of 0 is 1")
else:
    for i in range(1,x+1):
        fact=fact*i
    print("The factorial of {} is {}".format(x,fact))
```

```
The factorial of 5 is 120
```

*args

With these we can use variable number of arguments as a input.

In [4]:

```
def func_a(*input):
    sum1=0
    for i in input:
        sum1=sum1+i
    print(sum1)

func_a(10,20,30,40)
func_a(1,2,3,4,5,6,7,8,9)
```

```
100
45
```

**kwargs

It is used when the keyword argument is of datatype dictionary and arguments are key-value pair

In [5]:

```
def func_b(**city):
    for i in city:
        print("The city name is {} for State {}"
              .format(i,city[i]))
func_b(Chennai="Karnataka",Mumbai="Maharashtra",
       Bangalore="Karnataka")
```

```
The city name is Chennai for State Karnataka
The city name is Mumbai for State Maharashtra
The city name is Bangalore for State Karnataka
```

lambda() functions

lambda() functions also called as anonymous functions and can be used with other user defined functions.

They return a single expression but can have multiple arguments

In [6]:

```
x=lambda a:a**5
print(x(5))
```

```
3125
```

In [7]:

```
x=lambda a,b,c:a*b*c
print(x(10,20,30))
```

```
6000
```

In [8]:

```
def list_apply(list_a,fun):
    list_b=[fun(a) for a in list_a]
    print(list_b)
list1=[10,20,30,40,50]
func=lambda x:x**3

list_apply(list1,func)
```

```
[1000, 8000, 27000, 64000, 125000]
```