# Python Day 2

Tuesday, September 19, 2023 9:06 AM

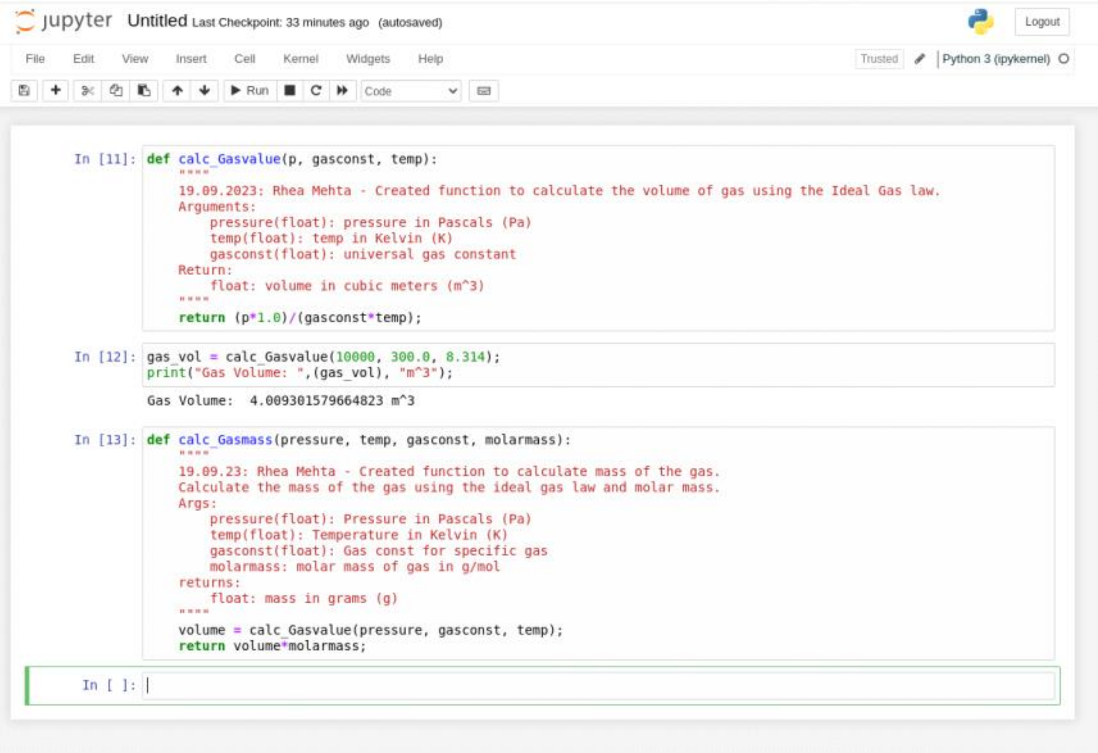
Function Composition:

```
def fun1:
  ---------------
  ---------------

def fun2:
  fun1:
  ---------------
  ---------------
```

Utility function: data preparation

Write a function that calculates the volume of gas, formula is:

(Pressure * 1.0)/gasconst * temp

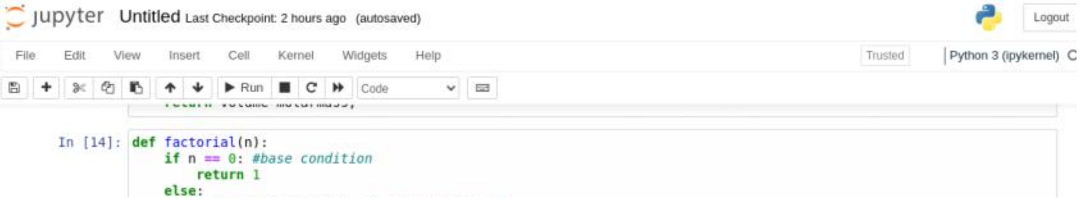


```
In [11]: def calc_Gasvalue(p, gasconst, temp):
             """
             19.09.2023: Rhea Mehta - Created function to calculate the volume of gas using the Ideal Gas law.
             Arguments:
                 pressure(float): pressure in Pascals (Pa)
                 temp(float): temp in Kelvin (K)
                 gasconst(float): universal gas constant
             Return:
                 float: volume in cubic meters (m^3)
             """
             return (p*1.0)/(gasconst*temp);

In [12]: gas_vol = calc_Gasvalue(10000, 300.0, 8.314);
         print("Gas Volume: ",(gas_vol), "m^3");

         Gas Volume:  4.009301579664823 m^3

In [13]: def calc_Gasmass(pressure, temp, gasconst, molarmass):
             """
             19.09.23: Rhea Mehta - Created function to calculate mass of the gas.
             Calculate the mass of the gas using the ideal gas law and molar mass.
             Args:
                 pressure(float): Pressure in Pascals (Pa)
                 temp(float): Temperature in Kelvin (K)
                 gasconst(float): Gas const for specific gas
                 molarmass: molar mass of gas in g/mol
             returns:
                 float: mass in grams (g)
             """
             volume = calc_Gasvalue(pressure, gasconst, temp);
             return volume*molarmass;

In [ ]:
```

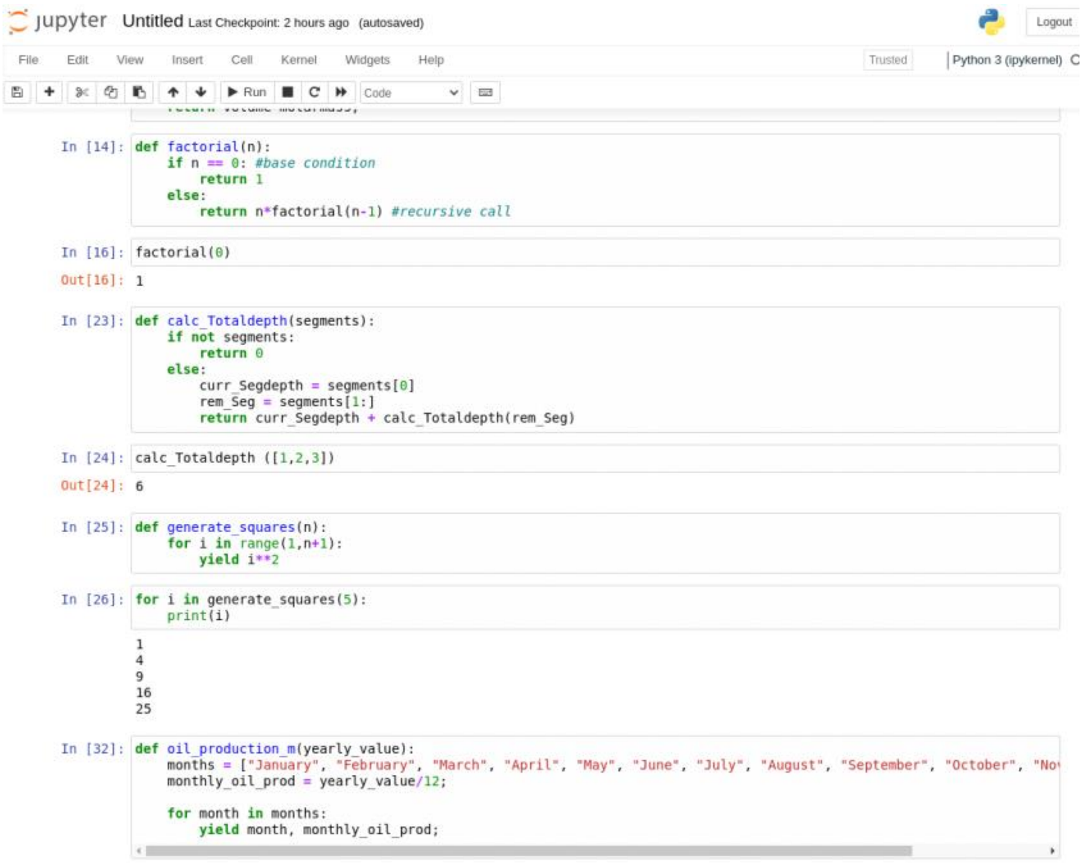
Generator function:

```
Def fun:
---------------
---------------
yield(a)
```

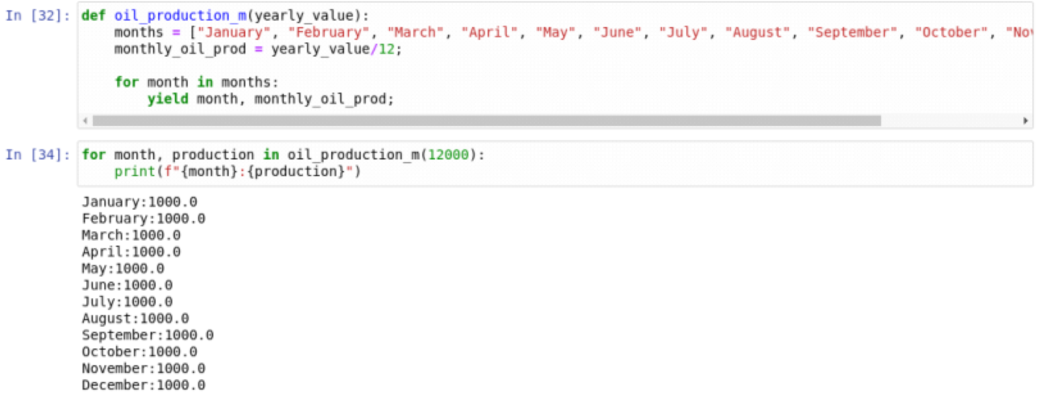
Yield is used with the generator function and is used to return multiple values (iterative) to the caller without terminating the program.



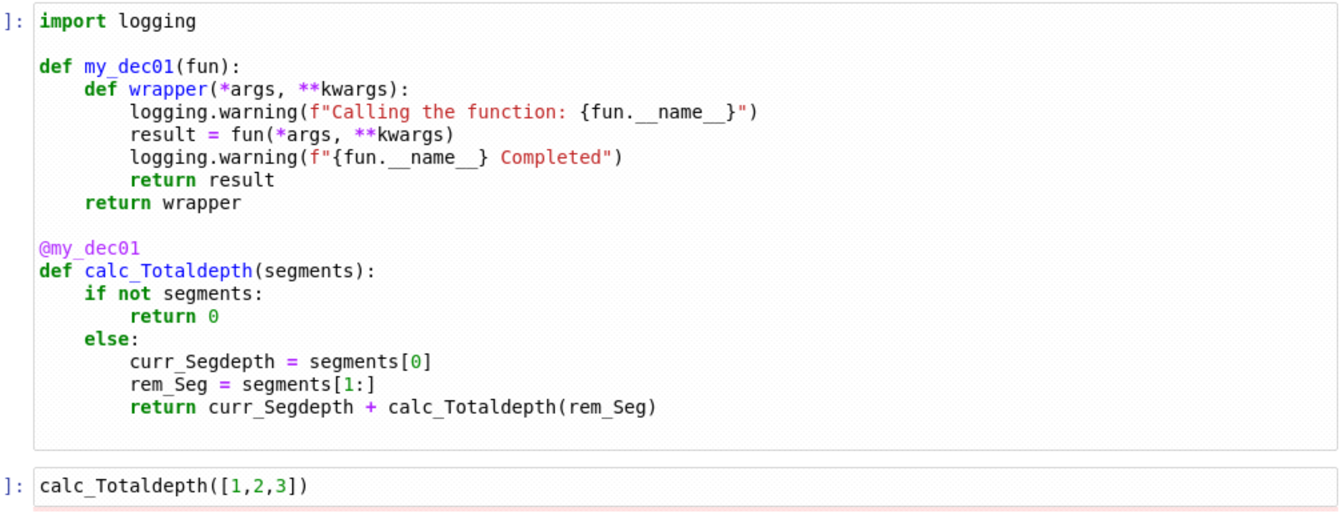
```
In [14]: def factorial(n):
             if n == 0: #base condition
                 return 1
             else:
```



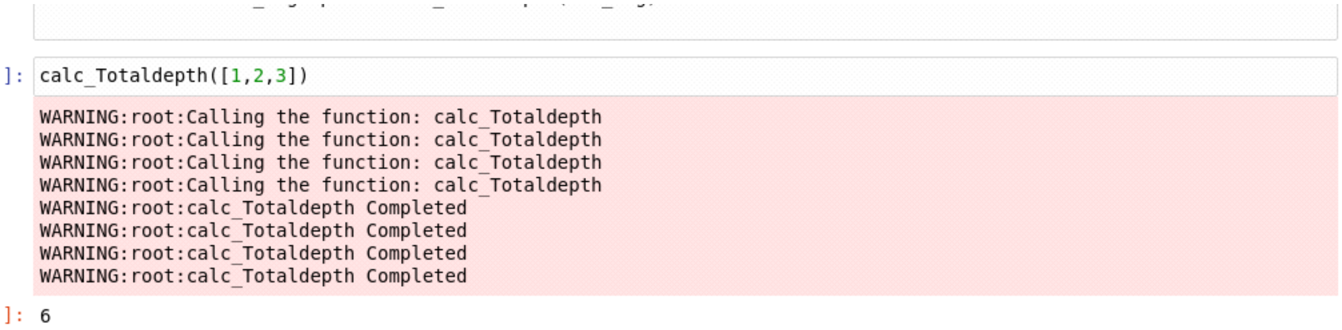
```
In [14]: def factorial(n):
             if n == 0: #base condition
                 return 1
             else:
                 return n*factorial(n-1) #recursive call

In [16]: factorial(0)

Out[16]: 1

In [23]: def calc_Totaldepth(segments):
             if not segments:
                 return 0
             else:
                 curr_Segdepth = segments[0]
                 rem_Seg = segments[1:]
                 return curr_Segdepth + calc_Totaldepth(rem_Seg)

In [24]: calc_Totaldepth ([1,2,3])

Out[24]: 6

In [25]: def generate_squares(n):
             for i in range(1,n+1):
                 yield i**2

In [26]: for i in generate_squares(5):
             print(i)

         1
         4
         9
         16
         25

In [32]: def oil_production_m(yearly_value):
             months = ["January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "No
             monthly_oil_prod = yearly_value/12;

             for month in months:
                 yield month, monthly_oil_prod;
```



```
In [32]: def oil_production_m(yearly_value):
             months = ["January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "No
             monthly_oil_prod = yearly_value/12;

             for month in months:
                 yield month, monthly_oil_prod;

In [34]: for month, production in oil_production_m(12000):
             print(f"{month}:{production}")

         January:1000.0
         February:1000.0
         March:1000.0
         April:1000.0
         May:1000.0
         June:1000.0
         July:1000.0
         August:1000.0
         September:1000.0
         October:1000.0
         November:1000.0
         December:1000.0
```



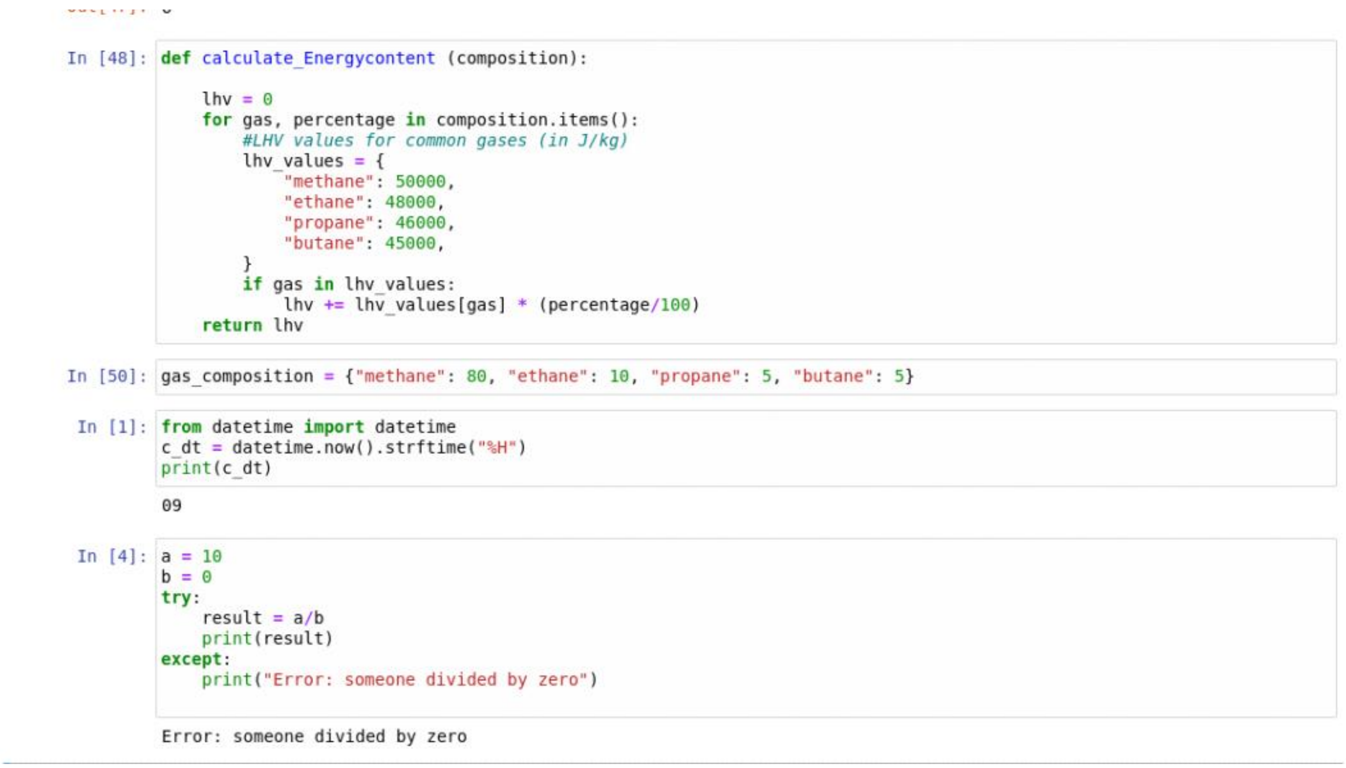
```
]: import logging

   def my_dec01(fun):
       def wrapper(*args, **kwargs):
           logging.warning(f"Calling the function: {fun.__name__}")
           result = fun(*args, **kwargs)
           logging.warning(f"{fun.__name__} Completed")
           return result
       return wrapper

   @my_dec01
   def calc_Totaldepth(segments):
       if not segments:
           return 0
       else:
           curr_Segdepth = segments[0]
           rem_Seg = segments[1:]
           return curr_Segdepth + calc_Totaldepth(rem_Seg)

]: calc_Totaldepth([1,2,3])
```



```
]: calc_Totaldepth([1,2,3])
```

```
WARNING:root:Calling the function: calc_Totaldepth
WARNING:root:Calling the function: calc_Totaldepth
WARNING:root:Calling the function: calc_Totaldepth
WARNING:root:Calling the function: calc_Totaldepth
WARNING:root:calc_Totaldepth Completed
WARNING:root:calc_Totaldepth Completed
WARNING:root:calc_Totaldepth Completed
WARNING:root:calc_Totaldepth Completed
```

```
]: 6
```



```
In [48]: def calculate_Energycontent (composition):

             lhv = 0
             for gas, percentage in composition.items():
                 #LHV values for common gases (in J/kg)
                 lhv_values = {
                     "methane": 50000,
                     "ethane": 48000,
                     "propane": 46000,
                     "butane": 45000,
                 }
                 if gas in lhv_values:
                     lhv += lhv_values[gas] * (percentage/100)
             return lhv
```

```
In [50]: gas_composition = {"methane": 80, "ethane": 10, "propane": 5, "butane": 5}
```

```
In [1]: from datetime import datetime
        c_dt = datetime.now().strftime("%H")
        print(c_dt)

        09
```

```
In [4]: a = 10
        b = 0
        try:
            result = a/b
            print(result)
        except:
            print("Error: someone divided by zero")

        Error: someone divided by zero
```