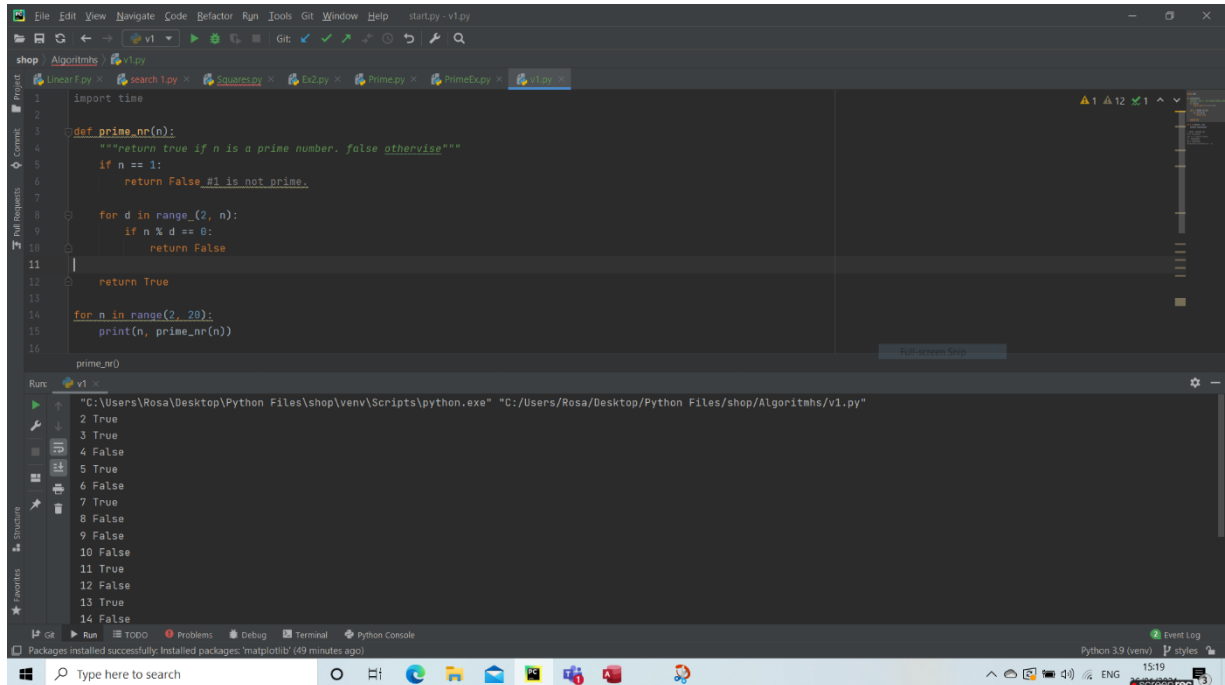


# Prime numbers

Here you can see the function that shows if the number is prime or not.



The screenshot shows a Python IDE with a file named `v1.py` open. The code defines a function `prime_nr(n)` that checks if a number `n` is prime. It returns `True` if `n` is prime and `False` otherwise. The function is tested for numbers from 2 to 20. The output shows the results for each number.

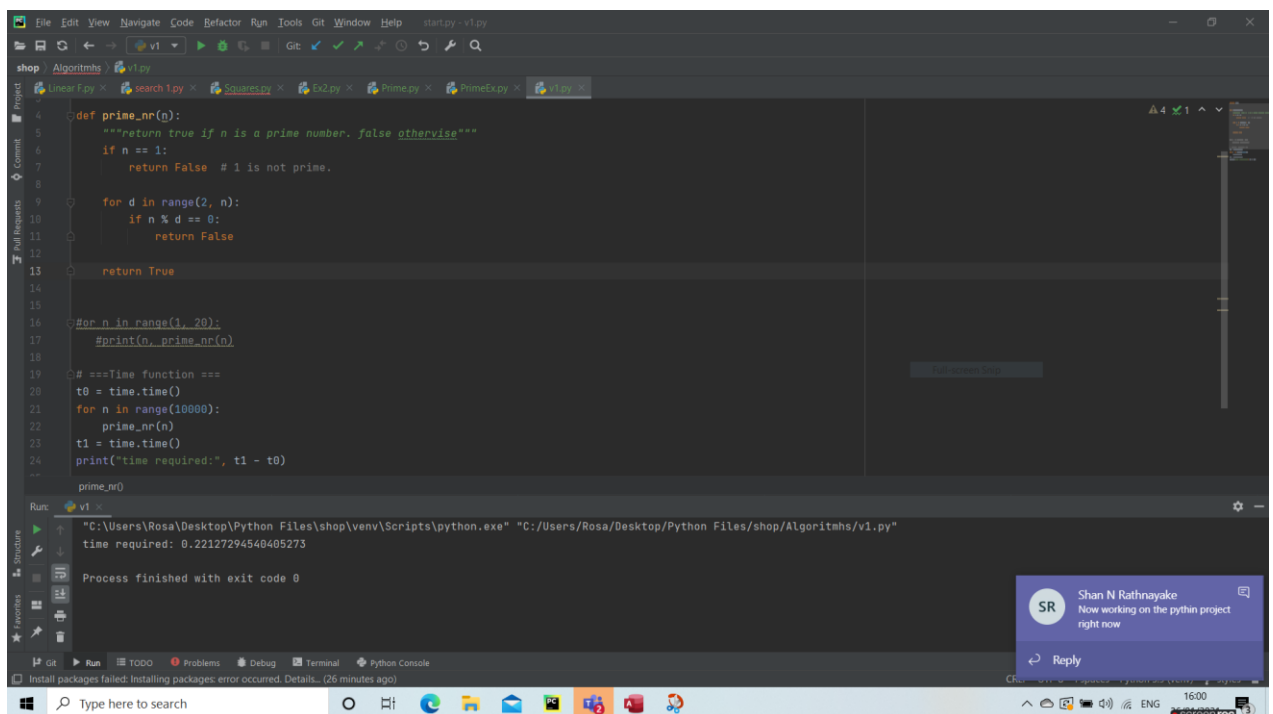
```
1 import time
2
3 def prime_nr(n):
4     """return true if n is a prime number. false otherwise"""
5     if n == 1:
6         return False #1 is not prime.
7
8     for d in range(2, n):
9         if n % d == 0:
10            return False
11
12    return True
13
14 for n in range(2, 20):
15    print(n, prime_nr(n))
16
17 prime_nr()
```

Run: `"C:\Users\Rosa\Desktop\Python Files\shop\venv\Scripts\python.exe" "C:/Users/Rosa/Desktop/Python Files/shop/Algorithms/v1.py"`

Output:

```
2 True
3 True
4 False
5 True
6 False
7 True
8 False
9 False
10 False
11 True
12 False
13 True
14 False
```

By adding the time function we can see how it affects the programs running time, depending on the number.



The screenshot shows a Python IDE with a file named `v1.py` open. The code defines a function `prime_nr(n)` that checks if a number `n` is prime. It returns `True` if `n` is prime and `False` otherwise. The function is tested for numbers from 1 to 20. The output shows the results for each number. The code also includes a timing function to measure the execution time of the prime number function for a range of numbers from 1 to 10000. The output shows the time required for the function to execute.

```
1 def prime_nr(n):
2     """return true if n is a prime number. false otherwise"""
3     if n == 1:
4         return False #1 is not prime.
5
6     for d in range(2, n):
7         if n % d == 0:
8             return False
9
10    return True
11
12 #for n in range(1, 20):
13 #    print(n, prime_nr(n))
14
15 # ===time function===
16 t0 = time.time()
17 for n in range(10000):
18     prime_nr(n)
19     t1 = time.time()
20     print("time required:", t1 - t0)
21
22 prime_nr()
```

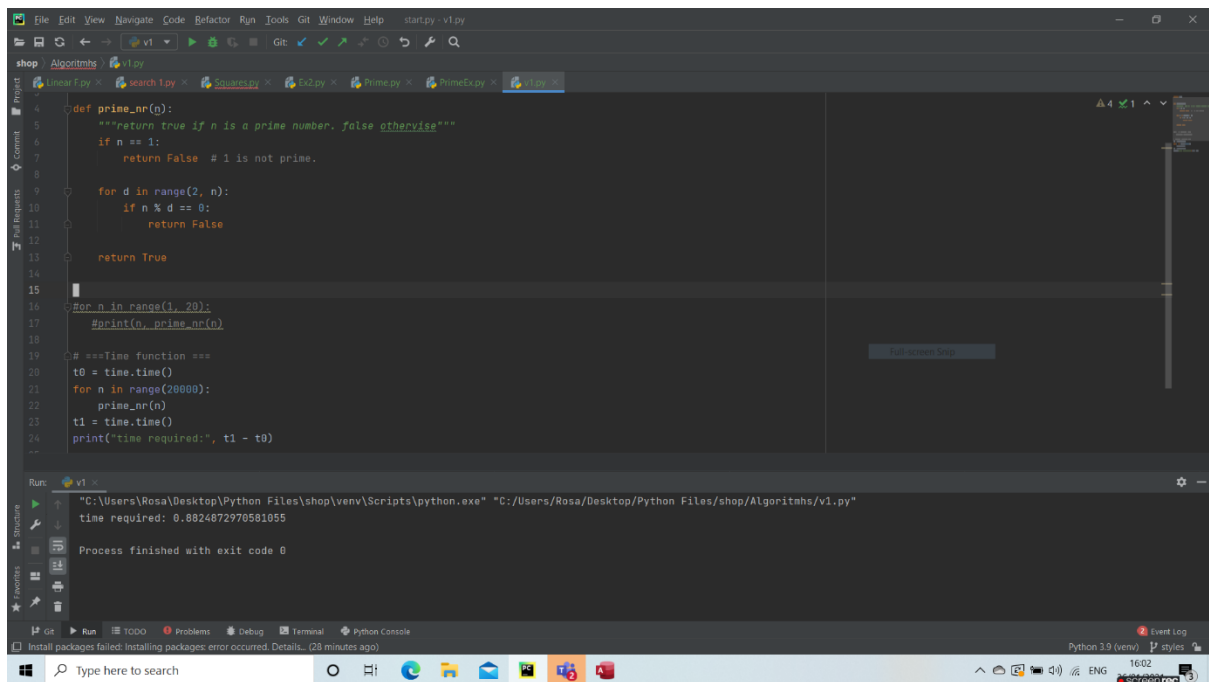
Run: `"C:\Users\Rosa\Desktop\Python Files\shop\venv\Scripts\python.exe" "C:/Users/Rosa/Desktop/Python Files/shop/Algorithms/v1.py"`

Output:

```
time required: 0.22127294540405273
```

Process finished with exit code 0

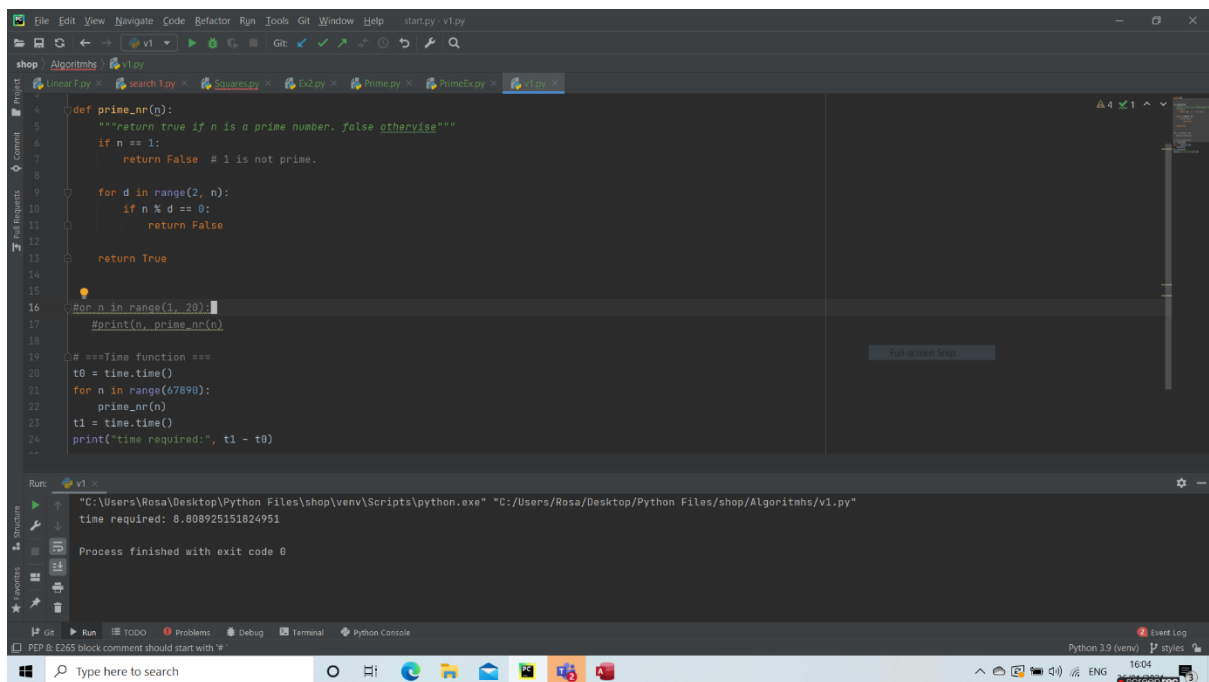
By increasing the range number, the running time grows significantly.



```
def prime_nr(n):  
    """return true if n is a prime number. false otherwise"""  
    if n == 1:  
        return False # 1 is not prime.  
    for d in range(2, n):  
        if n % d == 0:  
            return False  
    return True  
  
for n in range(1, 20):  
    #print(n, prime_nr(n))  
  
# ===Time function ===  
t0 = time.time()  
for n in range(20000):  
    prime_nr(n)  
t1 = time.time()  
print("time required:", t1 - t0)
```

Run: v1 x  
"C:\Users\Rosa\Desktop\Python Files\shop\venv\Scripts\python.exe" "C:\Users\Rosa\Desktop\Python Files\shop\Algorithms\v1.py"  
time required: 0.8824872970581055  
Process finished with exit code 0

As you can see, this function is not the best, as in bigger number cases, the running time is very high.



```
def prime_nr(n):  
    """return true if n is a prime number. false otherwise"""  
    if n == 1:  
        return False # 1 is not prime.  
    for d in range(2, n):  
        if n % d == 0:  
            return False  
    return True  
  
for n in range(1, 20):  
    #print(n, prime_nr(n))  
  
# ===Time function ===  
t0 = time.time()  
for n in range(67890):  
    prime_nr(n)  
t1 = time.time()  
print("time required:", t1 - t0)
```

Run: v1 x  
"C:\Users\Rosa\Desktop\Python Files\shop\venv\Scripts\python.exe" "C:\Users\Rosa\Desktop\Python Files\shop\Algorithms\v1.py"  
time required: 8.808925151824951  
Process finished with exit code 0

As you can see, my code is not very efficient, as it takes a very long time to run it.