Lab 1 Exercise

In [11]:
```
# Find the all factors of x using a loop and the operator %
# % means find remainder, for example 10 % 2 = 0; 10% 3 = 1
x = 52633
for i in range(x+1):
# your code here
    if (x % (i+1) == 0):
        print(i+1)
```

```
1
7
73
103
511
721
7519
52633
```

In [12]:
```
# Write a function that prints all factors of the given parameter x
def print_factors(x):
    factors = []
    for i in range(1, x + 1):
        if x % i == 0:
            factors.append(i)
    print("Factors of", x, "are:", factors)

print_factors(52633)
```

```
Factors of 52633 are: [1, 7, 73, 103, 511, 721, 7519, 52633]
```

In [13]:
```
# Write a program that be able to find all factors of the numbers in the list l
l = [52633, 8137, 1024, 999]
# your code here
for num in l:
    print_factors(num)
```

```
Factors of 52633 are: [1, 7, 73, 103, 511, 721, 7519, 52633]
Factors of 8137 are: [1, 79, 103, 8137]
Factors of 1024 are: [1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]
Factors of 999 are: [1, 3, 9, 27, 37, 111, 333, 999]
```

Lab 2 Exercise

In [15]:
```
import json
import requests
site="https://api.npoint.io/2b57052af2060e84dc86"
# Write the functions convert_number and replace_number here
# Follow the logic below.
# Trying to load JSON into text

def convert_number(lst):
    # Convert all elements (except the first one) into numbers and return as a list
    return [int(i) for i in lst[1:]]

def replace_number(number_list, being_replace, to_replace):
    # Replace all occurrences of 'being_replace' with 'to_replace' in the list
    return [to_replace if i == being_replace else i for i in number_list]

r = requests.get(site)
print(r.json())
text = r.json()['users']
# Debug
for i in text:
    print("parse " + str(i))
# call the function convert_number
# convert all elements (except the first one) into number and return it as a list
y = convert_number(text[0])

print("y")
print(y)
# call the function replace_number
# replace all number 1 by the number 10 in the function
z = replace_number(number_list = y, being_replace = 1, to_replace =10)
print("z")
print(z)
sum = 0

for i in z:
    sum = sum + i
    print("sum = " + str(sum) + "; i =" + str(i))
```

```
print ("Total = " + str(sum))
```

```
{'name': 'blogger', 'users': [['admins', '1', '2', '3'], ['editors', '4', '5', '6']]}
parse ['admins', '1', '2', '3']
parse ['editors', '4', '5', '6']
y
[1, 2, 3]
z
[10, 2, 3]
sum = 10; i =10
sum = 12; i =2
sum = 15; i =3
Total = 15
```

Write ups

1. What is the purpose of return in Python?

The return statement is used to end the execution of a function and return a value to the caller. When a function is called, it may perform certain computations and produce a result. The return statement allows the function to send this result back to the code that called the function.

2. Where can we define a Python function parameter? How can we use it?

In Python, function parameters are defined within the parentheses of a function's declaration.

As an example:

In [4]:
```
def greet(name):
    print(f"Hello, {name}!")

# Calling the function with a specific argument
greet("Christopher")  # Output: Hello, Alice!
```

```
Hello, Christopher!
```

3. Write a Python function that can reverse a string.

In [7]:
```
def rev_string(str):
    return str[::-1]
```

In [8]:
```
rev_string("Christopher")
```

Out[8]:
```
'rehpotsirhC'
```

4. What is a conflict in Git? What is the cause of such conflict?

In Git, a conflict occurs when there are conflicting changes made to the same part of a file or files within a repository. Conflicts typically happen during a process called "merging" when Git tries to combine changes from different branches, possibly due to the same code are being edited by different parties, accidentally or deliberately amend, add or delete some lines, or moved their location of line.

5. How can we resolve a conflict?

To resolve a conflict in Git, the user may need to review the conflicting file(s) and decide which changes are kept or removed. Git would indicate the conflicting regions by conflict markers (<<<<<<<, =======, and >>>>>>>). The contents between <<<<<<< and ======= are the changes made by the user while the contents between ======= and >>>>>>> are the changes from the remote repository. After the user decides the desired changes and removes the conflict markers, he/she should commit the changes and push them to the remote repository. Then, conflicts are resolved.

```
<<<<<<< HEAD
# Changes made in the current branch
=======
# Changes made in the incoming branch
>>>>>>> branch-name
```

6. What practices can be done to avoid having a conflict?

To avoid conflict in git, the below can be done:

1. Frequent Pulls: Regularly pull changes from the remote repository to keep your local copy up-to-date.

2. Better communication: Coordinate with your team to avoid working on the same files concurrently.
3. Branching Strategy: Isolate your work in feature branches when working on the same product, making it easier to manage conflicts during merges.

In [ ]: