Ifs, loops, and function homework

1. A function to reverse a string

Write and test a function that reverses a string entered by a user. This function will have one input value (a string) and one output value (also a string).

Test your function on, among other things, Napoleon's quote 'able was i ere i saw elba'

In [7]:
```
Nap = "able was i ere i saw elba"
print(Nap)
```

```
able was i ere i saw elba
```

Optional challenge: run the above on "race car" and then fix the resulting string.

In [6]:
```
reversednap = Nap[::-1]
print(reversednap)
```

```
able was i ere i saw elba
```

2. Determine if a number is prime

Write some code to test whether a number is prime or not, a prime number being an integer that is evenly divisible only by 1 and itself.

Hint: another way to think about a prime number is that, if the smallest number (other than 1) that divides evenly into a number *is* that number, than the number is a prime.

The easiest solution involves one `while` loop and one `if` test.

In [36]:
```
def is_prime(num):
    if num <= 1:
        return False

    for i in range(2, int(num**0.5) + 1):
        if num % i == 0:
            return False

    return True

number = 4
if is_prime(number):
    print(f"{number} is a prime number.")
else:
    print(f"{number} is not a prime number.")
```

```
4 is not a prime number.
```

3. Find the first 10 primes

Extend your code above to find the first 10 prime numbers. This will involve wrapping your existing code in another "outer" loop.

In [43]:
```
def is_prime(num):
    if num <= 1:
        return False

    for i in range(2, int(num**0.5) + 1):
        if num % i == 0:
            return False

    return True

number = 15
if is_prime(number):
    print(f"{number} is a prime number.")
else:
    print(f"{number} is not a prime number.")

count_primes = 0
num_to_check = 2

while count_primes < 10:
    if is_prime(num_to_check):
        print(num_to_check, end=" ")
        count_primes += 1
    num_to_check += 1

print()
```

```
15 is not a prime number.
2 3 5 7 11 13 17 19 23 29
```

4. Make a function to compute the first n primes

Functionalize (is that a word?) your above code. A user should be able to call your code with one integer argument and get a list back containing that number of primes. Make sure your function handles inputs of an incorrect type gracefully. You should also warn the user if they enter a really big number (which could take a long time...), and give them the option of either bailing or entering a different number.

In [44]:
```
def find_primes(num_primes):
    prime_list = []
    count_primes = 0
    num_to_check = 2

    while count_primes < num_primes:
        if is_prime(num_to_check):
            prime_list.append(num_to_check)
            count_primes += 1
        num_to_check += 1

    return prime_list
```

```python
#To test this function they can do
num_primes = 10
prime_numbers = find_primes(num_primes)
print(prime_numbers)
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
```