Define a function that takes an integer argument and returns a logical value true or false depending on if the integer is a prime.

Per Wikipedia, a prime number ( or a prime ) is a natural number greater than 1 that has no positive divisors other than 1 and itself.

## **Requirements**

* You can assume you will be given an integer input.
* You can not assume that the integer will be only positive. You may be given negative numbers as well ( or 0 ).
* NOTE on performance: There are no fancy optimizations required, but still *the* most trivial solutions might time out. Numbers go up to 2^31 ( or similar, depending on language ). Looping all the way up to n, or n/2, will be too slow.

def is\_prime(num):

if num <= 1:

return False

for i in range(2, num):

if num % i == 0:

return False

else:

return True

#Code wars didn’t want my code because it wasn’t “optimized”

#Here’s an optimized version to review later

import math # Import the math module for the isqrt function.

def is\_prime(num):

# Check if the input is less than or equal to 1. If so, it's not prime.

if num <= 1:

return False

# Check if the input is equal to 2. If so, it's prime.

if num == 2:

return True

# Check if the input is even (except for 2). If so, it's not prime.

if num % 2 == 0:

return False

# Calculate the maximum possible divisor to check, which is the square root of the input.

max\_divisor = math.isqrt(num) + 1

# Iterate through odd divisors from 3 up to the square root of the input.

for divisor in range(3, max\_divisor, 2):

# If the input is divisible by the current divisor, it's not prime.

if num % divisor == 0:

return False

# If none of the divisors evenly divides the input, it's prime.

return True

Explanation step by step:

# We import the math module to use the isqrt function, which calculates the integer square root of a number efficiently.

# The function first checks if the input number num is less than or equal to 1. If num is 1 or less, it returns False because prime numbers must be greater than 1.

# Next, it checks if num is equal to 2. If num is 2, it returns True because 2 is a prime number.

# Then, it checks if num is even (except for 2) by using the modulo operator (%). If num is even, it returns False because even numbers (other than 2) cannot be prime.

# It calculates the max\_divisor as the integer square root of num plus 1. This is the highest possible divisor we need to check to determine if num is prime.

# The loop iterates through odd divisors starting from 3 up to max\_divisor. We start from 3 because we've already handled the case of 2. The step size of 2 ensures that we only check odd divisors because even divisors greater than 2 cannot divide num.

# Inside the loop, it checks if the current divisor evenly divides num. If it does, the function returns False because num is not prime.

# If none of the divisors within the specified range evenly divides num, the function returns True, indicating that num is prime.

# This optimized code efficiently checks for prime numbers, avoiding unnecessary iterations and improving performance for larger numbers.

def is\_prime(num):

if num <= 1:

return False

if num <= 3:

return True

if num % 2 == 0 or num % 3 == 0:

return False

i = 5

while i \* i <= num:

if num % i == 0 or num % (i + 2) == 0:

return False

i += 6

return True

# The given code defines a Python function called is\_prime that checks whether a given integer num is a prime number or not. Let's break down how this code works step by step:

# Initial Checks:

# The function starts with two initial checks:

# If num is less than or equal to 1, it immediately returns False since prime numbers must be greater than 1.

# If num is less than or equal to 3, it immediately returns True because 2 and 3 are prime numbers.

# Divisibility by 2 and 3:

# The code checks if num is divisible by 2 or 3. If num is divisible by either of these two numbers, it returns False because prime numbers cannot be evenly divisible by numbers other than 1 and themselves.

# Loop for Divisibility:

# The code then enters a loop that starts with i initialized to 5. This loop continues as long as i \* i is less than or equal to num.

# Within the loop, it checks two conditions:

# If num is divisible by i, it returns False.

# If num is divisible by (i + 2), it returns False.

# The loop increments i by 6 in each iteration. This is because, after checking divisibility for i, there's no need to check for divisibility for i + 1 or i + 4 because they cannot be prime (they are either divisible by 2 or 3).

# Prime Number Check:

# If none of the conditions within the loop are met, the function returns True, indicating that num is a prime number.

# This code is an implementation of a common optimization technique for checking prime numbers, known as the "6k +/- 1" rule. It only checks for divisibility by numbers of the form 6k ± 1, where k is an integer. This rule reduces the number of potential divisors that need to be checked, making the primality test more efficient for larger numbers compared to a brute-force approach of checking divisibility by all numbers up to the square root of num.