Lab Manual

Practical and Skills Development

CERTIFICATE

THE ASSIGNMENT ENTERED IN THIS REPORT HAVE BEEN SATISFACTORILY PERFORMED BY

Registration No	: 25BAI10734
Name of Student	: Atharav Balaji Khonde
Course Name	: Introduction to Problem Solving and Programming
Course Code	: CSE1021
School Name	: VIT Bhopal (cse artificial intelligence and Machine learning)
Slot	: B11+B12+B13
Class ID	: BL2025260100796
Semester	: FALL 2025/26

Course Faculty Name : Dr. Hemraj S. Lamkuche

Signature:

VIT BHOPAL

Practical Index

S. No.	Title of Practical	Date of Submission	Signature of Faculty
1			
2			
3			
4			
5			
6	factorial(n)	05-10-2025	
7	is_palindrome(n)	05-10-2025	
8	mean_of_digits(n)	05-10-2025	
9	digital_root(n)	05-10-2025	
10	is_abundant(n)	05-10-2025	
11			
12			
13			

14			
15			

Practical No: 6

Date: 05-10-2025

TITLE: factorial(n)

AIM/OBJECTIVE(s): to make a function factorial(n) that calculates the factorial of a non-negative integer n (n!).

METHODOLOGY & TOOL USED:

Defining a function factorial(n) then calculating execution time and memory allocation then taking user input and applying function factorial(n) on it

Tools used are import time and import tracemalloc

BRIEF DESCRIPTION:

First we Import the tracemalloc and time, which is a tool used to trace memory allocations and, which is a tool for time-related tasks, including measuring code execution time respectively.

Then we start measuring time and memory allocation using command start = time.time() and tracemalloc.start()

Then we defines a function named factorial(n) that takes one argument, n.

Then we initializes a variable named factorial to 1. This variable will store the calculated factorial value.

for x in range(1, n+1) we starts a for loop that iterates through all integers from 1 up to n.

factorial *= x: Multiplies the current value of factorial by the loop variable x. This is the core logic for calculating the factorial.

Then we print final value of factorial using print(factorial)

We record the end time again by end = time.time()

Then we print executed time by print(end-start).

current, peak = tracemalloc.get_traced_memory(): Retrieves the current and peak memory usage recorded by tracemalloc.

print(f"Current memory usage is {current }; Peak was {peak}"): Prints the measured memory usage. current is the memory currently in use, while peak is the highest memory usage that occurred during execution.

n = int(input("enter your number")): Prompts the user to "enter your number," reads the input, converts it to an integer, and stores it in the variable n.

factorial(n): Calls the factorial function, passing the user-provided number n as an argument to perform the calculation.

RESULTS ACHIEVED: a function factorial(n) that calculates the factorial of a non-negative integer n (n!) is made

DIFFICULTY FACED BY STUDENT:

Multiplying all the result factorial including n itself using factorial *= x

Initialising a variable named factorial = 1 which will store all the factorial values to multiply

Other difficulties are are defining function and from where and how to start a loop

Syntax and common errors

SKILLS ACHIEVED:

Able to design a function to calculate factorial of number and getting to use the loop with more variation and getting more logic building ideas and encountering more errors

```python
import tracemalloc
import time
start = time.time()
tracemalloc.start()
def factorial(n):
    factorial = 1
    for x in range(1,n+1):
        factorial*= x
    print(factorial)
end = time.time()
print(end-start)
current, peak = tracemalloc.get_traced_memory()
print(f"Current memory usage is {current }; Peak was {peak}")

n = int(input("enter your number"))
factorial(n)
```

```
0.0005495548248291016
Current memory usage is 2245; Peak was 29186
enter your number5
120
```

Practical No: 7

Date: 05-10-2025

TITLE: is_palindrome(n)

AIM/OBJECTIVE(s): Write a function is_palindrome(n) that checks if a number reads the same forwards and backwards.

METHODOLOGY & TOOL USED:

Defining a function is_palindrome(n) then calculating execution time and memory allocation then taking user input and applying function factorial(n) on it

Tools used are import time and import tracemalloc

BRIEF DESCRIPTION:

import tracemalloc: Imports the tracemalloc module, which is a tool for tracing Python's memory allocations.

start = time.time(): Records the current time in seconds since the "epoch" (the point where time begins) and stores it in the start variable.

tracemalloc.start(): Starts tracking Python's memory allocations from this point.

def is_palindrome(n): Defines a function named is_palindrome that takes one argument, n.

n_str = str(n): Converts the input number n into a string and assigns it to nstr.

n_str_reverse = str[::-1]: The line attempts to reverse the string

return n_str == n_str reversed: It attempts to compare the original string to the reversed one.

end = time.time(): Records the time again after the code has run.

print(end - start): Prints the total execution time by calculating the difference between the end and start timestamps.

Then we Prints the memory usage using current, peak = tracemalloc.get_traced_memory(): Retrieves the current and peak memory usage recorded by tracemalloc.

print(f"Current memory usage is {current }; Peak was {peak}"): Prints the measured memory usage. current is the memory currently in use, while peak is the highest memory usage that occurred during execution

num = input(): Prompts the user to enter a value from the console and stores it as a string in the num variable.

if is_palindrome(num):: Calls the is_palindrome function with the user's input.

print(...): Prints whether the number is a palindrome or not based on the function's return value, using an f-string for formatted output.

RESULTS ACHIEVED: a function is_palindrome(n) that checks if a number reads the same forwards and backwards is made.

DIFFICULTY FACED BY STUDENT:

Writing command to reverse the string and taking input from user and converting it to string and reversing it and comparing and many errors

SKILLS ACHIEVED:

Able to reverse string, getting to know more about strings and reversing

Able to build a function is_palindrome(n)

Encountering more errors

```python
import tracemalloc
import time
start = time.time()
tracemalloc.start()
def is_palindrome(n):
  n_str = str(n)
  n_str_reversed = n_str[::-1]
  return n_str == n_str_reversed
end = time.time()
print(end-start)
current, peak = tracemalloc.get_traced_memory()
print(f"Current memory usage is {current }; Peak was {peak}")

num = input()
if is_palindrome(num):
  print(f"{num} is a palindrome")
else:
  print(f"{num} is not a palindrome")
```

```
0.0005502700805664062
Current memory usage is 590078; Peak was 4910747
45
45 is not a palindrome
```

Practical No: 8

Date: 05-10-2025

TITLE: mean_of_digits(n)

AIM/OBJECTIVE(s): Write a function mean_of_digits(n) that returns the average of all digits in a number.

METHODOLOGY & TOOL USED:

Defining a function mean_of_digits(n) then calculating execution time and memory allocation then taking user input and applying function factorial(n) on it

Tools used are import time and import tracemalloc

BRIEF DESCRIPTION:

import tracemalloc and import time: These lines import the necessary libraries for tracking memory allocation (tracemalloc) and measuring time (time).

start = time.time(): Records the starting time before the calculation begins.

tracemalloc.start(): Starts tracking memory allocations.

def mean_of_digits(n)**:**: Defines a function called mean_of_digits that takes an integer n as input.

n_str = str(n): Converts the input number n to a string so that each digit can be iterated over.

sum_of_digits = 0 and count_of_digits = 0: Initializes variables to store the sum of the digits and the count of the digits.

for digit in n_str:: This loop iterates through each character (digit) in the string n_str.

sum_of_digits += int(digit): Converts the current digit character back to an integer and adds it to sum_of_digits.

count_of_digits += 1: Increments the count of digits for each digit processed.

if count_of_digits == 0:: Checks if the number of digits is zero (this would happen if the input was an empty string, although the code converts an integer to a string, so this is unlikely with a valid integer input).

return 0: If the count is zero, it returns 0 to avoid division by zero.

else: return sum_of_digits / count_of_digits: If there are digits, it calculates and returns the mean (sum divided by count).

end = time.time(): Records the ending time after the calculation is complete.

print(end-start): Calculates and prints the difference between the end and start times, showing the execution time of the code.

current, peak = tracemalloc.get_traced_memory(): Gets the current and peak memory usage tracked by tracemalloc.

print(f'Current memory usage is {current }; Peak was {peak}"): Prints the current and peak memory usage.

num1 = int(input()): Prompts the user to enter a number and stores it as an integer in the variable num1.

print(f'The mean of digits in {num1} is: {mean_of_digits(num1)}"): Calls the mean_of_digits function with the user-provided number and prints the result in a formatted string.

RESULTS ACHIEVED: a function mean_of_digits(n) that returns the average of all digits in a number is acheived

DIFFICULTY FACED BY STUDENT:

Separating the digits of number and converting them to integer and adding them and errors

SKILLS ACHIEVED: able to separate digits of a number

And able to build a function mean_of_digits(n)

Encountering more errors and building more logic and concepts

```
import tracemalloc
import time
start = time.time()
tracemalloc.start()

def mean_of_digits(n):
  n_str = str(n)
  sum_of_digits = 0
  count_of_digits = 0
  for digit in n_str:
    sum_of_digits += int(digit)
    count_of_digits += 1
  if count_of_digits == 0:
    return 0
  else:
    return sum_of_digits / count_of_digits

end = time.time()
print(end-start)
current, peak = tracemalloc.get_traced_memory()
print(f"Current memory usage is {current }; Peak was {peak}")

num1 = int(input())
print(f"The mean of digits in {num1} is: {mean_of_digits(num1)}")
```

```
0.0011720657348632812
Current memory usage is 3063038; Peak was 4910747
67
The mean of digits in 67 is: 6.5
```

Practical No: 9

Date: 05-10-2025

TITLE: digital_root(n)

AIM/OBJECTIVE(s): Write a function digital_root(n) that repeatedly sums the digits of a number until a single digit is obtained.

METHODOLOGY & TOOL USED:

Defining a function digital_root(n) then calculating execution time and memory allocation then taking user input and applying function factorial(n) on it

Tools used are import time and import tracemalloc

BRIEF DESCRIPTION:

import tracemalloc and import time: These lines import the necessary libraries for tracking memory allocation (tracemalloc) and measuring time (time).

start = time.time(): Records the starting time before the calculation begins.

tracemalloc.start(): Starts tracking memory allocations.

def digital_root(n):: Defines a function called digital_root that takes an integer n as input.

while n > 9:: This loop continues as long as the number n is greater than 9. The digital root is a single digit (0-9).

sum_of_digits = 0: Initializes a variable sum_of_digits to 0 for each iteration of the outer while loop.

for digit in str(n):: Converts the current number n to a string and iterates through each character (digit).

sum_of_digits += int(digit): Converts the current digit character back to an integer and adds it to sum_of_digits.

n = sum_of_digits: After summing the digits of the current number, the result becomes the new value of n. The while loop then checks if this new n is still greater than 9.

return n: Once the while loop finishes (meaning n is 9 or less), the function returns the final single-digit value of n, which is the digital root.

end = time.time(): Records the ending time after the calculation is complete.

print(end-start): Calculates and prints the difference between the end and start times, showing the execution time of the code.

current, peak = tracemalloc.get_traced_memory(): Gets the current and peak memory usage tracked by tracemalloc.

print(f"Current memory usage is {current }; Peak was {peak}"): Prints the current and peak memory usage.

num1 = int(input()): Prompts the user to enter a number and stores it as an integer in the variable num1.

print(f"The digital root of {num1} is: {digital_root(num1)}"): Calls the digital_root function with the user-provided number and prints the result in a formatted string.

RESULTS ACHIEVED: a function digital_root(n) that repeatedly sums the digits of a number until a single digit is obtained is achevied

DIFFICULTY FACED BY STUDENT:

Converting the current number n to a string and iterates through each character (digit).

Initializing a variable sum_of_digits to 0 for each iteration of the outer while loop

And errors

SKILLS ACHIEVED:

Converting the current number n to a string and iterates through each character (digit).

Able to build function digital_root_(n)

More logic building and variation of iterating through each charecter

```python
import tracemalloc
import time
start = time.time()
tracemalloc.start()
def digital_root(n):
    while n > 9:
        sum_of_digits = 0
        for digit in str(n):
            sum_of_digits += int(digit)
        n = sum_of_digits
    return n
end = time.time()
print(end-start)
current, peak = tracemalloc.get_traced_memory()
print(f"Current memory usage is {current }; Peak was {peak}")


num1 = int(input())
print(f"The digital root of {num1} is: {digital_root(num1)}")
```

```
0.0010056495666503906
Current memory usage is 1313233; Peak was 4910747
45
The digital root of 45 is: 9
```

Practical No: 10

Date: 05-10-2025

TITLE: is_abundant(n)

AIM/OBJECTIVE(s): Write a function is_abundant(n) that returns True if the sum of proper divisors of n is greater than n.

METHODOLOGY & TOOL USED:

Defining a function is_abundant(n) then calculating execution time and memory allocation then taking user input and applying function factorial(n) on it

Tools used are import time and import tracemalloc

BRIEF DESCRIPTION:

import tracemalloc and import time: These lines import the necessary libraries for tracking memory allocation (tracemalloc) and measuring time (time).

start = time.time(): Records the starting time before the calculation begins.

tracemalloc.start(): Starts tracking memory allocations.

def is_abundant(n):: Defines a function called is_abundant that takes an integer n as input.

proper_divisors_sum = 0: Initializes a variable proper_divisors_sum to 0. This will store the sum of the proper divisors of n.

for i in range(1, n // 2 + 1):: This loop iterates through possible divisors from 1 up to half of n (inclusive). Proper divisors are divisors of a number, excluding the number itself. We only need to check up to n // 2 because any divisor larger than n // 2 would have a corresponding divisor smaller than n // 2 (except for n itself).

if n % i == 0:: Checks if i is a divisor of n. The modulo operator (%) returns the remainder of a division. If the remainder is 0, i is a divisor.

proper_divisors_sum += i: If i is a divisor, it is added to proper_divisors_sum.

return proper_divisors_sum > n: After checking all possible proper divisors, the function returns True if the sum of the proper divisors is greater than n (meaning it's an abundant number), and False otherwise.

end = time.time(): Records the ending time after the calculation is complete.

print(end-start): Calculates and prints the difference between the end and start times, showing the execution time of the code.

current, peak = tracemalloc.get_traced_memory(): Gets the current and peak memory usage tracked by tracemalloc.

print(f"Current memory usage is {current }; Peak was {peak}"): Prints the current and peak memory usage.

num1 = int(input()): Prompts the user to enter a number and stores it as an integer in the variable num1.

if is_abundant(num1):: Calls the is_abundant function with the user-provided number num1.

print(f"{num1} is an abundant number"): If the function returns True, it prints that the number is abundant.

else: print(f"{num1} is not an abundant number"): If the function returns False, it prints that the number is not abundant

RESULTS ACHIEVED: a function is_abundant(n) that returns True if the sum of proper divisors of n is greater than n is achevied

DIFFICULTY FACED BY STUDENT:

Initializing a variable proper_divisors_sum to 0 which will store the sum of the proper divisors of n.

Iterating loop through possible divisors from 1 up to half of n (inclusive). Proper divisors are divisors of a number, excluding the number itself

Proper divisor sum and Returning proper divisor sum

SKILLS ACHIEVED:

Variation of iterating loop and learning about return function

Encountering more errors

Able to build function is_abundant(n)

```
import tracemalloc
import time

start = time.time()
tracemalloc.start()

def is_abundant(n):
  proper_divisors_sum = 0
  for i in range(1, n // 2 + 1):
    if n % i == 0:
      proper_divisors_sum += i
  return proper_divisors_sum > n

end = time.time()
print(end-start)
current, peak = tracemalloc.get_traced_memory()
print(f"Current memory usage is {current }; Peak was {peak}")

num1 =int(input())
if is_abundant(num1):
  print(f"{num1} is an abundant number")
else:
  print(f"{num1} is not an abundant number")
```

```
0.0006339550018310547
Current memory usage is 915074; Peak was 4910747
45
45 is not an abundant number
```