jupyter task 2 Last Checkpoint: 5 minutes ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

[1]:
```python
import math

def is_prime(n):
    if n <= 1:
        return False
    if n == 2:
        return True
    if n % 2 == 0:
        return False
    for i in range(3, int(math.sqrt(n)) + 1, 2):
        if n % i == 0:
            return False
    return True
```

[2]:
```python
num = int(input("Enter a number to check if it's prime: "))
print("Prime:", is_prime(num))
```

```
Enter a number to check if it's prime:  37
Prime: True
```

[ ]:

jupyter task 2 Last Checkpoint: 9 minutes ago

File Edit View Run Kernel Settings Help Truste

Code JupyterLab Python 3 (ipykernel)

[4]:
```python
# Sum of Digits

# Input
n = int(input("Enter a number: "))

# Convert number to string and sum digits
sum_of_digits = sum(int(digit) for digit in str(n))

# Output
print("Sum of digits:", sum_of_digits)
```

```
Enter a number:  348
Sum of digits: 15
```

[5]:
```python
#LCM AND GCD
import math

# Input
a = int(input("Enter first number (a): "))
b = int(input("Enter second number (b): "))

# Calculate GCD
gcd = math.gcd(a, b)

# Calculate LCM using the formula: |a * b| / GCD
lcm = abs(a * b) // gcd

# Output
print("GCD:", gcd)
print("LCM:", lcm)
```

```
Enter first number (a):  18
Enter second number (b):  24
GCD: 6
LCM: 72
```

jupyter task 2 Last Checkpoint: 16 minutes ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

[7]:
```python
# LIST REVERSAL
# Input: Define the list
original_list = [1, 2, 3, 4, 5]

# Reversing the list using slicing (no built-in reverse())
reversed_list = original_list[::-1]

# Output
print("Original list:", original_list)
print("Reversed list:", reversed_list)
```

```
Original list: [1, 2, 3, 4, 5]
Reversed list: [5, 4, 3, 2, 1]
```

jupyter task 2 Last Checkpoint: 18 minutes ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

[8]:
```python
# SORT A LIST
# Input
numbers = [5, 2, 9, 1, 5, 6]

# Bubble sort
n = len(numbers)
for i in range(n):
    for j in range(0, n - i - 1):
        if numbers[j] > numbers[j + 1]:
            # Swap
            numbers[j], numbers[j + 1] = numbers[j + 1], numbers[j]

# Output
print("Sorted list:", numbers)
```

```
Sorted list: [1, 2, 5, 5, 6, 9]
```

jupyter task 2 Last Checkpoint: 19 minutes ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

[9]:
```python
#REMOVE DUPLICATES
# Input list
numbers = [1, 2, 2, 3, 4, 4, 5]

# Remove duplicates using set
unique_list = list(set(numbers))

# Optional: sort if order matters
unique_list.sort()

# Output
print("List with duplicates removed:", unique_list)
```

```
List with duplicates removed: [1, 2, 3, 4, 5]
```

jupyter task 2 Last Checkpoint: 21 minutes ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

[10]:
```python
#STRING LENGTH
# Input
text = input("Enter a string: ")

# Count characters manually
length = 0
for char in text:
    length += 1

# Output
print("Length of the string:", length)
```

```
Enter a string:  I LIVE IN MUMBAI
Length of the string: 16
```

jupyter task 2 Last Checkpoint: 24 minutes ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

[11]:
```python
# Count Vowels and Consonants
# Input
text = input("Enter a string: ")

# Define vowels
vowels = set("aeiouAEIOU")

# Initialize counters
vowel_count = 0
consonant_count = 0

# Loop through each character
for char in text:
    if char.isalpha():  # Check if character is a letter
        if char in vowels:
            vowel_count += 1
        else:
            consonant_count += 1

# Output
print("Vowels:", vowel_count)
print("Consonants:", consonant_count)
```

```
Enter a string:  do you like icecream?
Vowels: 9
Consonants: 8
```

jupyter task 2 Last Checkpoint: 40 minutes ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

[14]:
```python
#maze generator and solver
import random
from collections import deque

# Maze constants
WALL = 1
PATH = 0
VISITED = 2
SOLUTION = 3

# Maze dimensions (must be odd for paths)
ROWS = 21
COLS = 21

# Directions: up, down, left, right
DIRS = [(-2, 0), (2, 0), (0, -2), (0, 2)]

def create_empty_maze(rows, cols):
    return [[WALL for _ in range(cols)] for _ in range(rows)]

def is_valid(r, c, maze):
    return 0 <= r < len(maze) and 0 <= c < len(maze[0])

def generate_maze_dfs(maze, r=1, c=1):
    maze[r][c] = PATH
    directions = DIRS[:]
    random.shuffle(directions)
    for dr, dc in directions:
        nr, nc = r + dr, c + dc
        if is_valid(nr, nc, maze) and maze[nr][nc] == WALL:
            maze[r + dr // 2][c + dc // 2] = PATH
            generate_maze_dfs(maze, nr, nc)
```

jupyter task 2 Last Checkpoint: 40 minutes ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

```python
def print_maze(maze):
    for row in maze:
        line = ""
        for cell in row:
            if cell == WALL:
                line += "#"
            elif cell == PATH:
                line += " "
            elif cell == VISITED:
                line += "."
            elif cell == SOLUTION:
                line += "*"
        print(line)

def solve_maze_dfs(maze, r, c, end):
    if (r, c) == end:
        maze[r][c] = SOLUTION
        return True
    if not is_valid(r, c, maze) or maze[r][c] != PATH:
        return False

    maze[r][c] = VISITED
    for dr, dc in [(-1,0), (1,0), (0,-1), (0,1)]:
        if solve_maze_dfs(maze, r+dr, c+dc, end):
            maze[r][c] = SOLUTION
            return True
    return False

def solve_maze_bfs(maze, start, end):
    queue = deque([start])
    prev = {start: None}
    while queue:
        r, c = queue.popleft()
        if (r, c) == end:
```

```python
            break
        for dr, dc in [(-1,0), (1,0), (0,-1), (0,1)]:
            nr, nc = r + dr, c + dc
            if is_valid(nr, nc, maze) and maze[nr][nc] == PATH and (nr, nc) not in prev:
                queue.append((nr, nc))
                prev[(nr, nc)] = (r, c)
    # Reconstruct path
    curr = end
    while curr and curr in prev:
        r, c = curr
        maze[r][c] = SOLUTION
        curr = prev[curr]

# Main function
maze = create_empty_maze(ROWS, COLS)
generate_maze_dfs(maze)

start = (1, 1)
end = (ROWS - 2, COLS - 2)

# Choose one of the solvers
# solve_maze_dfs(maze, *start, end)
solve_maze_bfs(maze, start, end)

print_maze(maze)
```

```python
print_maze(maze)
```

```
╔═══════════════════╗
╚*****╝ ╚           ╗
═══*╝ ╚ ╚ ═══════ ╚
╚*****╝ ╚ ╚       ╚ ╚
╚*═══ ╚ ═══ ═══ ═══
╚*╚ ╚   ╚     ╚   ╚
╚*╚ ╚ ╚ ═══ ═══ ╚ ╚
╚*╚   ╚     ***╚ ╚ ╚
╚*═══════════*╚*═══ ╚
╚*****═****═*╚*****╚
═══*╚*═══*╚*═══*╚ ╚
╚***╚*╚***╚*╚ ╚***╚
╚*╚*╚*╚*═══*╚ ═══*
╚*╚***╚*╚   *╚   ╚*
╚*═══*╚*╚  *╚═══ ╚*
╚***╚***╚*╚ ***╚ ╚*
═══*╚*═══*╚ ═══*╚ ╚*
╚***╚*╚ ╚*╚     ╚*╚***
╚*═══*  ╚*═══*╚*╚═══
╚*****╚   *******╚***
╚═══════════════════╝
```