Task 1: Create a DataFrame

```
import pandas as pd
import numpy as np
from functools import reduce

# Task 1
data = {'Name': ['Alice', 'Bob', 'Charlie', 'David'],
        'Age': [25, 30, 35, 28],
        'City': ['New York', 'San Francisco', 'Los Angeles', 'Chicago']}
df = pd.DataFrame(data)
print(df)
```

```
      Name  Age           City
0    Alice   25       New York
1      Bob   30  San Francisco
2  Charlie   35    Los Angeles
3    David   28        Chicago
```

Task 2: Row and Column Manipulation

```
df_dropped = df.drop(columns=['City'])
print(df_dropped)
```

```
      Name  Age
0    Alice   25
1      Bob   30
2  Charlie   35
3    David   28
```

** Task 3: Handling Null Values**

```
df_nulls = pd.DataFrame({
    'Name': ['Eve', None, 'Frank'],
    'Age': [22, None, 29],
    'City': [None, 'Toronto', 'Vancouver']
})

# Fill nulls: strings with 'Unknown', numbers with 0
df_nulls_filled = df_nulls.fillna({'Name': 'Unknown', 'Age': 0, 'City': 'Unknown'})
print(df_nulls_filled)
```

```
      Name   Age       City
0      Eve  22.0    Unknown
1  Unknown   0.0    Toronto
2    Frank  29.0  Vancouver
```

** Task 4: GroupBy and Describe**

```
df_group = pd.DataFrame({
    'Category': ['A', 'B', 'A', 'B', 'A', 'C'],
    'Value': [10, 20, 15, 25, 30, 35]
})
grouped_stats = df_group.groupby('Category')['Value'].describe()
print(grouped_stats)
```

```
          count       mean        std   min    25%   50%    75%   max
Category
A           3.0  18.333333  10.408330  10.0  12.50  15.0  22.50  30.0
B           2.0  22.500000   3.535534  20.0  21.25  22.5  23.75  25.0
C           1.0  35.000000        NaN  35.0  35.00  35.0  35.00  35.0
```

** Task 5: Concatenation and Merging**

```
df1 = pd.DataFrame({'A': [1, 2], 'B': [3, 4]})
df2 = pd.DataFrame({'A': [5, 6], 'B': [7, 8]})
df3 = pd.DataFrame({'C': [9, 10, 11, 12], 'D': [13, 14, 15, 16]})

# Concatenate vertically
df_concat = pd.concat([df1, df2], ignore_index=True)

# Merge horizontally
df_merged = pd.concat([df_concat, df3], axis=1)
print(df_merged)
```

```
   A  B   C   D
0  1  3   9  13
1  2  4  10  14
2  5  7  11  15
3  6  8  12  16
```

** Task 6: Tuples and Sets**

```
fruits = ('apple', 'banana', 'cherry')
numbers = {1, 2, 3, 4, 5}
```

```
# Try to add elements
try:
    fruits.append('orange')  # Error: tuples are immutable
except AttributeError as e:
    print(f"Tuple error: {e}")

numbers.add(6)
print(numbers)
```

```
Tuple error: 'tuple' object has no attribute 'append'
{1, 2, 3, 4, 5, 6}
```

Task 7: Dictionaries

```
scores = {'Aisha': 85, 'Zaid': 90, 'Sara': 78}
scores['Zaid'] = 95  # Update score
scores['Omar'] = 88  # Add new student
print(scores)
```

```
{'Aisha': 85, 'Zaid': 95, 'Sara': 78, 'Omar': 88}
```

** Task 8: Functions and Lambda**

```
def square(x):
    return x * x

square_lambda = lambda x: x * x

print(square(3), square_lambda(4))
```

```
9 16
```

** Task 9: Iterators and Generators**

```
# Iterator class
class EvenNumbers:
    def __init__(self, max_count=5):
        self.count = 0
        self.num = 0
        self.max_count = max_count

    def __iter__(self):
        return self

    def __next__(self):
        if self.count >= self.max_count:
            raise StopIteration
        self.num += 2
        self.count += 1
        return self.num

for even in EvenNumbers():
    print(even, end=" ")

print()

# Generator function
def even_gen(n=5):
    num = 2
    for _ in range(n):
        yield num
        num += 2

for even in even_gen():
    print(even, end=" ")
```

```
2 4 6 8 10
2 4 6 8 10
```

Task 10: Map, Reduce, and Filter

```
numbers = [1, 2, 3, 4, 5]

squared = list(map(lambda x: x ** 2, numbers))
product = reduce(lambda x, y: x * y, numbers)
evens = list(filter(lambda x: x % 2 == 0, numbers))

print("Squared:", squared)
print("Product:", product)
print("Evens:", evens)
```

```
Squared: [1, 4, 9, 16, 25]
Product: 120
Evens: [2, 4]
```

Task 11: Object-Oriented Programming

```
class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width

    def area(self):
        return self.length * self.width

    def perimeter(self):
        return 2 * (self.length + self.width)

r1 = Rectangle(4, 5)
r2 = Rectangle(6, 3)

print(f"Area r1: {r1.area()}, Perimeter r1: {r1.perimeter()}")
print(f"Area r2: {r2.area()}, Perimeter r2: {r2.perimeter()}")
```

```
Area r1: 20, Perimeter r1: 18
Area r2: 18, Perimeter r2: 18
```

Task 12: Pandas Data Analysis

```
df_employees = pd.DataFrame({
    'Name': ['John', 'Jane', 'Bob', 'Alice', 'Charlie'],
    'Department': ['IT', 'HR', 'IT', 'Finance', 'HR'],
    'Salary': [55000, 65000, 70000, 60000, 58000]
})

# Average salary by department
print(df_employees.groupby('Department')['Salary'].mean())

# Employees with salary > 60000
print(df_employees[df_employees['Salary'] > 60000]['Name'])

# Add Bonus column
df_employees['Bonus'] = df_employees['Salary'] * 0.10
print(df_employees)
```

```
Department
Finance    60000.0
HR         61500.0
IT         62500.0
Name: Salary, dtype: float64
1    Jane
2     Bob
Name: Name, dtype: object
      Name Department  Salary   Bonus
0     John         IT   55000  5500.0
1     Jane         HR   65000  6500.0
2      Bob         IT   70000  7000.0
3    Alice    Finance   60000  6000.0
4  Charlie         HR   58000  5800.0
```