```
        print(f"{function_args[0].__name__}(x, y) =", printable)
```

This program prints the following to the console:

```
x = [ 1  5 10 20 30]
y = [ 2  7  5 10 30]
f(x) = [ 13  49 139 469 999]
g(x) = [1.708 1.92  1.296 1.833 1.976]
h(x, y) = [0.743 1.946 1.609 2.303 3.401]
F(x, y) = [0. 0. 2. 2. 1.]
G(x, y) = [  2  10 125 500  60]
```

**Problem 3.6** DN Write a program that graphs each of the following functions over the specified domain:

a. $f(x) = \tanh(4 \sin x)$ for $x \in [-5, 8]$

b. $g(x) = \sin \sqrt{x}$ for $x \in [0, 100]$

c. $h(x) = \begin{cases} 0 & \text{if } x < 0 \\ e^{-x} \sin(2\pi x) & \text{otherwise} \end{cases}$ for $x \in [-2, 6]$

**Solution 3.6** DN

```
"""Solution to Chapter 3 problem DN"""
import numpy as np
import matplotlib.pyplot as plt
```

**Introduction** This program defines several mathematical functions as vectorized functions that can handle NumPy array inputs and plots them over the given domain using Matplotlib.

**Define Mathematical Functions** Define $f(x) = x^2 + 3x + 9$:

```
def f(x: np.ndarray) -> np.ndarray:
    return np.tanh(4 * np.sin(x))
```

Define $g(x) = 1 + \sin^2 x$:

```
def g(x: np.ndarray) -> np.ndarray:
    return np.sin(np.sqrt(x))
```

Define $h(x, y) = e^{-3x} + \ln y$:

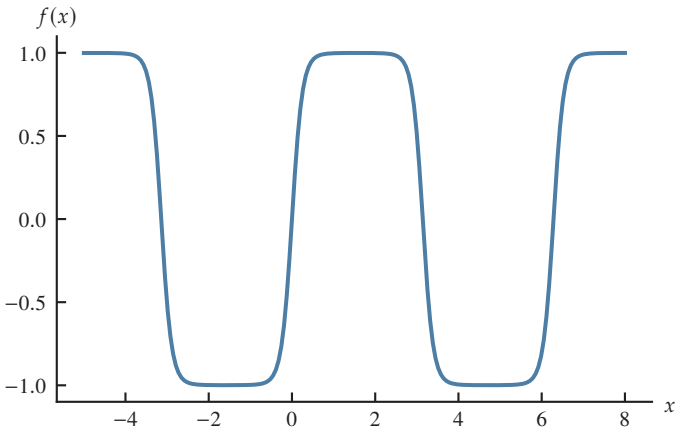
```
def h(x: np.ndarray) -> np.ndarray:
    return np.where(x >= 0, np.exp(-x) * np.sin(2 * np.pi * x), 0)
```

**Plotting** Define a plotting function:

```
def plotter(fig, fun, limits, labels):
    x = np.linspace(limits[0], limits[1], 201)
    fig.gca().plot(x, fun(x))
    fig.gca().set_xlabel(labels[0])
    fig.gca().set_ylabel(labels[1])
    return fig
```

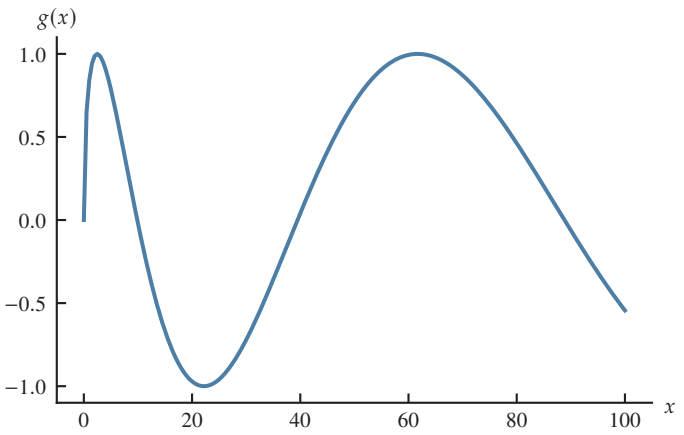
Plot $f(x)$:

```
fig, ax = plt.subplots()
plotter(fig, fun=f, limits=(-5, 8), labels=("$x$", "$f(x)$"))
```



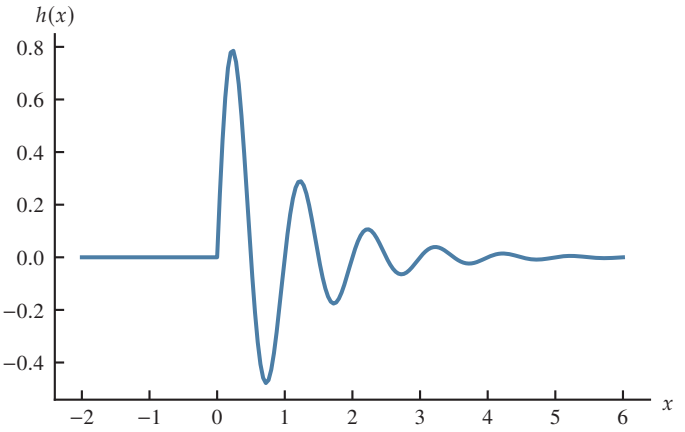

Plot $g(x)$:

```
fig, ax = plt.subplots()
plotter(fig, fun=g, limits=(0, 100), labels=("$x$", "$g(x)$"))
```



Plot $h(x)$:

```
fig, ax = plt.subplots()
plotter(fig, fun=h, limits=(-2, 6), labels=("$x$", "$h(x)$"))
```




```
plt.show()
```

**Problem 3.7** WF Write a program that loads and plots ideal gas data with the `engcom.data.ideal_gas()` function in the following way:

a. The data it loads is over the volume domain: $V \in [0.1, 2.1]$ m$^3$
b. The data it loads has 3 temperatures: $V = 300, 400, 500$ K
c. It plots in a single graphic $P$ versus $V$ for each of the three temperatures
d. Each data point should be marked with a dot •
e. Sequential data points should be connected by straight lines
f. Each plot should be labeled with its corresponding temperature, either next to the plot or in a legend

**Solution 3.7** WF

```
"""Solution to Chapter 3 problem WF"""
import numpy as np
import matplotlib.pyplot as plt
import engcom.data
```

**Introduction** This program loads and plots ideal gas data with the `ideal_gas()` function from the `engcom.data` module.

**Load the Data**

```
d = engcom.data.ideal_gas(
    V=np.linspace(0.1, 2.1, 16),  # (m^3) Volume values
    T=np.array([300, 400, 500]),  # (K) Temperature values
)
```

**Plot** Now `d` is a dictionary with the following key–value pairs:

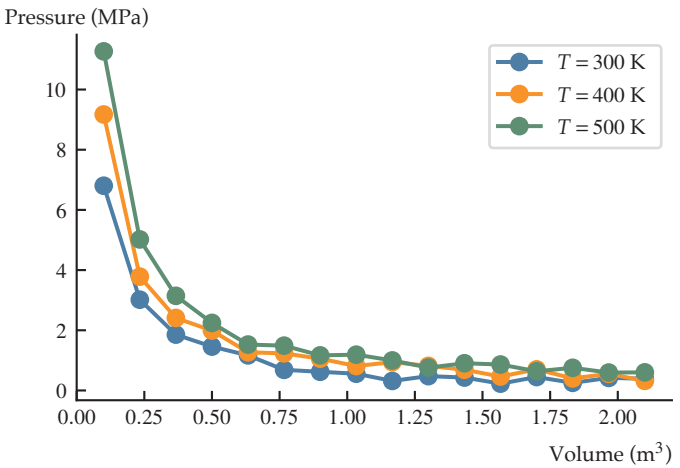
- `"volume"`: $V_{16\times1}$ ($m^3$)
- `"temperature"`: $T_{1\times3}$ (K)
- `"pressure"`: $P_{16\times3}$ (Pa)

We would like to plot $P$ versus $V$ for each of the 3 temperatures $T$; that is, plot a sequence of pairs $(P_i, V_i)$ for each $T_j$. The following code loops through the temperatures and plots to the same axes object:

```
fig, ax = plt.subplots()
for j, Tj in enumerate(d["temperature"].flatten()):
    x = d["volume"]  # (m^3)
    y = d["pressure"][:, j] / 1e6  # (MPa)
    ax.plot(x, y, marker="o", label=f"$T = {Tj}$ K")
```

Finally, we label the axes and display the figure with the following code:

```
ax.set_xlabel("Volume (m$^3$)")
ax.set_ylabel("Pressure (MPa)")
ax.legend()  # Show labels in legend
plt.show()
```




**Problem 3.8** Y1 Use the data from problem 3.7 to write a program that meets the following requirements:

a. It loads the pressure-volume-temperature data from problem 3.7.
b. It estimates the work $W$ done by the gas for each of the three values of temperatures via the integral equation

$$W = -\int_{0.1}^{2.1} P(V)\, dV.$$

*Note*: An integral can be estimated from discrete data via the trapezoidal rule, which can be executed with NumPy's `np.trapz()` function.
c. It generates a bar chart comparing the three values of work (one for each temperature).

**Solution 3.8** Y1

```
"""Solution to Chapter 3 problem Y1"""
import numpy as np
import matplotlib.pyplot as plt
import engcom.data
```

**Introduction** This program loads ideal gas data with the engcom.data.ideal_gas() function. It computes the work done by the gas over the given volume. Finally, it charts the work for each temperature.

**Load the Data** Load the pressure-volume-temperature data as follows:

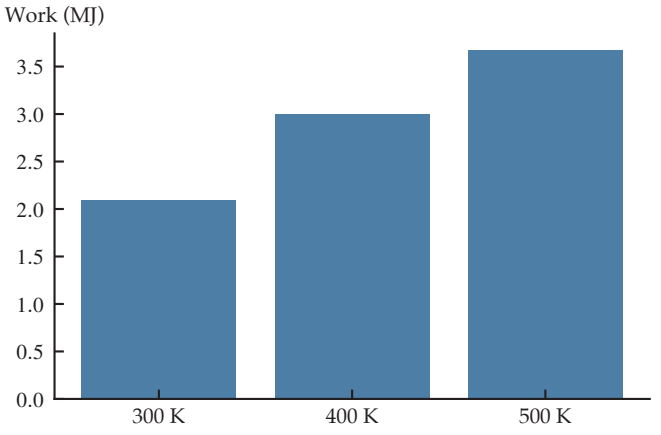
```
d = engcom.data.ideal_gas(
    V=np.linspace(0.1, 2.1, 16),  # (m^3) Volume values
    T=np.array([300, 400, 500]),  # (K) Temperature values
)
```

**Define and Compute the Work** Define a function to compute the work and apply the function.

```
def W(P: np.ndarray, V: np.ndarray, axis: int = -1) -> float:
    """Use trapezoidal integration to estimate the work from P(V)"""
    return np.trapz(P, V, axis=0)
work = W(P=d["pressure"], V=d["volume"])
```

**Plot** Create a bar chart of work for each temperature

```
x = np.arange(len(work))
labels = np.char.add(d["temperature"].flatten().astype(dtype=str), " K")
fig, ax = plt.subplots()
ax.bar(x, work / 1e6)
ax.set_xticks(x, labels=labels)
ax.set_ylabel("Work (MJ)")
plt.show()
```


Work (MJ)
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
300 K
400 K
500 K