Exercise 1 - Part 1: Introduction to Python

October 11, 2024

Gestione Energetica ed Automazione negli Edifici (GEAE) A.A. 2024/2025

1 Exercise 1 - Part 1: Introduction to Python

1.1 Variables and Data Types

Assigning values to variables. This code demonstrates how to assign values to different data types in Python. We have the following variables: - `temperature`: an integer representing temperature in Celsius. - `pressure`: a float representing atmospheric pressure in bars. - `gas`: a string representing the type of gas (in this case, "Methane"). - `system_operational`: a boolean indicating whether the system is operational.

[1]:
```
temperature = 25  # integer
pressure = 1.01325  # float (floating-point number)
gas = "Methane"  # string
system_operational = True  # boolean

print("The temperature is:", temperature, "°C")
print("The pressure is:", pressure, "bar")
print("The type of gas is:", gas)
print("Is the system operational?", system_operational)
```

```
The temperature is: 25 °C
The pressure is: 1.01325 bar
The type of gas is: Methane
Is the system operational? True
```

1.2 Mathematical Operations

Define mass and velocity

[2]:
```
mass = 10  # kg
velocity = 5  # m/s
```

Calculate the kinetic energy of the object.

```
[3]: kinetic_energy = 0.5 * mass * velocity ** 2
```

1.3 User Input

Ask the user to enter the height of an object

```
[4]: mass = 10  # kg (ensure mass is defined for potential energy calculation)
     height = float(input("Enter the height in meters: "))
     gravity = 9.81  # m/s^2
     potential_energy = mass * gravity * height
```

```
[5]: print("The potential energy of the object is:", potential_energy, "Joules")
```

```
The potential energy of the object is: 784.8000000000001 Joules
```

1.4 Conditional Statements

Conditional statements help control the flow of a program based on conditions. For example, checking the state of water based on its temperature:

```
[6]: water_temperature = 30

     if water_temperature <= 0:
         state = "solid (ice)"
     elif water_temperature >= 100:
         state = "gaseous (steam)"
     else:
         state = "liquid"

     print(f"At {water_temperature} °C, water is in the {state} state.")
```

```
At 30 °C, water is in the liquid state.
```

This code allows you to determine the state of water based on temperature input, which can be useful in automation systems to determine operational states.

1.5 Lists

A list is an ordered and mutable collection of items. Lists are useful for storing multiple related values. For example:

```
[7]: daily_temperatures = [20, 22, 24, 23, 25, 21, 19]  # list of integers
     print("Daily temperatures:", daily_temperatures)
```

```
Daily temperatures: [20, 22, 24, 23, 25, 21, 19]
```

You can access elements using their index, modify elements, and add new elements. Lists are very flexible and are often used to handle collections of data such as sensor readings or control commands.

Accessing elements of a list:

```
[8]: first_temperature = daily_temperatures[0]   # index 0
     print("The first temperature is:", first_temperature, "°C")
```

```
The first temperature is: 20 °C
```

Modifying elements of a list

```
[9]: daily_temperatures[2] = 26
     print("Updated list of temperatures:", daily_temperatures)
```

```
Updated list of temperatures: [20, 22, 26, 23, 25, 21, 19]
```

Adding an element to the list

```
[10]: daily_temperatures.append(18)
      print("List of temperatures after adding a value:", daily_temperatures)
```

```
List of temperatures after adding a value: [20, 22, 26, 23, 25, 21, 19, 18]
```

1.6 Dictionaries

Dictionaries store key-value pairs and are extremely useful for representing data that is inherently connected. For instance:

```
[11]: gas_properties = {
          "Methane": {"Molar Mass": 16.04, "Boiling Point": -161.5},
          "Ethane": {"Molar Mass": 30.07, "Boiling Point": -88.6},
          "Propane": {"Molar Mass": 44.10, "Boiling Point": -42.1}
      }

      print("Properties of Methane:", gas_properties["Methane"])
```

```
Properties of Methane: {'Molar Mass': 16.04, 'Boiling Point': -161.5}
```

Adding a new element to the dictionary

```
[12]: gas_properties["Butane"] = {"Molar Mass": 58.12, "Boiling Point": -0.5}
      print("Updated gas properties dictionary:", gas_properties)
```

```
Updated gas properties dictionary: {'Methane': {'Molar Mass': 16.04, 'Boiling
Point': -161.5}, 'Ethane': {'Molar Mass': 30.07, 'Boiling Point': -88.6},
'Propane': {'Molar Mass': 44.1, 'Boiling Point': -42.1}, 'Butane': {'Molar
Mass': 58.12, 'Boiling Point': -0.5}}
```

1.6.1 Loops

Loops are used to iterate over collections of data, such as lists or dictionaries. A `for` loop is particularly useful when you need to perform repetitive tasks.

For example, iterating over a list:

[13]:
```
print("List of daily temperatures:")
for temp in daily_temperatures:
    print(temp, "°C")
```

```
List of daily temperatures:
20 °C
22 °C
26 °C
23 °C
25 °C
21 °C
19 °C
18 °C
```

You can also iterate through a dictionary to access both keys and values, which is useful for displaying information or performing calculations on multiple components.

[14]:
```
print("List of gas properties:")
for gas, properties in gas_properties.items():
    print(f"{gas}: Molar Mass = {properties['Molar Mass']} g/mol, Boiling Point␣
 ↪= {properties['Boiling Point']} °C")
```

```
List of gas properties:
Methane: Molar Mass = 16.04 g/mol, Boiling Point = -161.5 °C
Ethane: Molar Mass = 30.07 g/mol, Boiling Point = -88.6 °C
Propane: Molar Mass = 44.1 g/mol, Boiling Point = -42.1 °C
Butane: Molar Mass = 58.12 g/mol, Boiling Point = -0.5 °C
```

1.7 Functions

Functions allow you to encapsulate code into reusable blocks. For example, defining a function to calculate specific heat capacity:

[15]:
```
def specific_heat_capacity(mass, heat_capacity):
    """
    Calculates the specific heat capacity.
    mass: mass of the object in kg
    heat_capacity: heat capacity in J/K
    """
    c = heat_capacity / mass
    return c

object_mass = 5  # kg
heat_capacity = 2500  # J/K

c = specific_heat_capacity(object_mass, heat_capacity)
print("The specific heat capacity of the object is:", c, "J/(kg·K)")
```

```
The specific heat capacity of the object is: 500.0 J/(kg·K)
```

This function takes mass, energy, and temperature change as arguments and returns the specific heat capacity. Functions make code more modular, readable, and reusable.

1.8 Libraries

Python provides a wide variety of built-in libraries to extend its capabilities. For example, the `math` library allows for complex mathematical operations:

[16]:
```python
import math

# Calculating the natural logarithm
number = 10
logarithm = math.log(number)
print("The natural logarithm of", number, "is:", logarithm)
```

```
The natural logarithm of 10 is: 2.302585092994046
```