Introduction to Python

Author: Nitesh Kumar

UPES
UNIVERSITY OF THE FUTURE

Email: nitesh.kumar@ddn.upes.ac.in

Python Basics from Scratch

Table of Contents

---

1. Introduction to Python

Python is a high-level, interpreted programming language known for its readability and simplicity. It is widely used in various fields, including web development, data analysis, artificial intelligence, and more.

Example Code

In [70]:
```
print("Hello, World!")
```

```
Hello, World!
```

Challenge

1. Print your name.
2. Print the result of 2 + 3.
3. Print a string that includes your favorite hobby.
4. Print the current year.
5. Print a simple math operation (e.g., 10 * 5).

In [9]:
```
a = 'Hello'
b = 'World !'
```

```
print(a + b)
```

```
HelloWorld !
```

In [ ]:

```
```

---

2. Variables and Data Types

Variables are used to store data values. Python has various data types, including integers, floats, strings, and booleans.

Example Code

In [76]:

```
# Integer
age = 25
# Float
height = 5.9
# String
name = "Alice"
# Boolean
is_student = True

print(age, height, name, is_student)
```

```
25 5.9 Alice True
```

Challenge

1. Create a variable for your age and print it.
2. Create a variable for your height and print it.
3. Create a variable for your favorite color and print it.
4. Create a boolean variable indicating if you like Python.
5. Print all the variables you created.

In [ ]:

```
```

In [ ]:

```
```

---

3. Basic Operators

Operators are used to perform operations on variables and values. Python supports arithmetic, comparison, and logical operators.

Example Code

In [82]:

```
a = 10
b = 5

# Arithmetic Operators
sum_result = a + b
difference = a - b
product = a * b
quotient = a / b
```

```
remainder = a % b
exponent = a ** b

print(sum_result, difference, product, quotient, remainder, exponent)
```

```
15 5 50 2.0 0 100000
```

Challenge

1. Calculate the remainder of 10 divided by 3.
2. Find the power of 2 raised to 5.
3. Subtract 15 from 30 and print the result.
4. Multiply 7 by 6 and print the result.
5. Divide 100 by 4 and print the result.

In [85]:
```
# calculating the volume and total surface area of a cylinder

r = 2.3
h = 5.5
pi = 3.1415

V = pi*r*r*h
A = 2*pi*r*(r+h)

print(A)
print(V)
```

```
112.71701999999999
91.40194249999999
```

3.1 USER INPUT

If you want to provide user input after running your code

```
r = input('Enter the radius of the cylinder=') #Automatically assume the str
r = float(r)

h = input('Enter the radius of the cylinder=') #Automatically assume the str
h = float(h)

pi = 3.1415

V = pi*r*r*h
A = 2*pi*r*(r+h)

print('\n')
print('Total surface area:', A)
print('Total Volume:', V)
```

In [11]:
```
r = input('Enter the radius of the cylinder=') #Automatically assume the str
r = float(r)

h = input('Enter the radius of the cylinder=') #Automatically assume the str
h = float(h)

pi = 3.1415

V = pi*r*r*h
A = 2*pi*r*(r+h)

print('\n') # ?
print('Total surface area:', A)
print('Total Volume:', V)
```

```
Total surface area: 376.98
Total Volume: 452.376
```

In [ ]:

```

```

print option (Using 'f' string)

`V = pi*r*r*h`

`print(f'The volume of the cylinder is {V} cm**3')`

- In the 'f' string we can choose the dtype and format of the output value
- `print(f'{V:0.2f})` means the floating point till 2 decimal points
- `print(f'{V:0f})` means the integer values
- `print(f'{V:.2e})` means the scientific notations and the first Mantissa is float till 2 decimal points

In [92]:

```
V = pi*r*r*h

print(f'The volume of the cylinder is {V:.0f} cm**3 \n')
print(f'The volume of the cylinder is {V:.2f} cm**3 \n')
print(f'The volume of the cylinder is {V:.2e} cm**3 \n')
```

```
The volume of the cylinder is 343523 cm**3

The volume of the cylinder is 343523.03 cm**3

The volume of the cylinder is 3.44e+05 cm**3
```

In [ ]:

```

```

---

4. Control Structures and loops

Control structures allow you to control the flow of your program. This includes conditional statements and loops.

Example Code

In [4]:

```
# If-Else Statement
number = 10
if number > 0:
    print("Positive number")
else:
    print("Negative number")

# For Loop
for i in range(5):
    print(i)
```

```
Positive number
0
1
2
3
4
```

Challenge

1. Write a program that checks if a number is even or odd.
2. Create a loop that prints numbers from 1 to 10.
3. Write a program that prints "Hello" 5 times.
4. Create a program that prints all numbers from 1 to 20 that are divisible by 3.
5. Write a program that prints the first 5 squares (1, 4, 9, 16, 25).

In [8]:
```
for i in range(11):
    if i % 2 == 0:
        print(i, 'is even')
    else:
        print(i, 'is odd')
```

```
0 is even
1 is odd
2 is even
3 is odd
4 is even
5 is odd
6 is even
7 is odd
8 is even
9 is odd
10 is even
```

In [ ]:
```
```

4.1 while loop

In [4]:
```
i = 1
while i < 6:
  print(i)
  i += 1
```

```
1
2
3
4
5
```

In [ ]:
```
```

In [ ]:
```
```

---

5. Functions

Functions are blocks of code that perform a specific task and can be reused.

Example Code

In [133…
```python
def greet(name):
    return f"Hello, {name}!"

print(greet("Alice"))
```

```
Hello, Alice!
```

In [17]:
```python
def sum(a, b):
    c = a + b
    return c

z = sum(5, 8)
print(z)
```

```
13
```

In [ ]:
```python
def is_prime(num):

    # code
    # check the divisibility by all numbers between 1 and num-1
    # if the num is not divisible by any number between 1 and num -1; it is a prime

    return result
```

In [ ]:

In [ ]:

Challenge

1. Write a function that takes two numbers and returns their sum.
2. Create a function that checks if a number is prime.
3. Write a function that returns the factorial of a number.
4. Create a function that takes a string and returns its length.
5. Write a function that converts Celsius to Fahrenheit.

In [ ]:

In [ ]:

---

6. Lists and Tuples

Lists and tuples are used to store multiple items in a single variable. Lists are mutable, while tuples are immutable.

Example Code

In [139…
```python
# List
fruits = ["apple", "banana", "cherry"]
fruits.append("orange")
print(fruits)

# Tuple
colors = ("red", "green", "blue")
print(colors)
```

```
['apple', 'banana', 'cherry', 'orange']
('red', 'green', 'blue')
```

```
In [3]: # Python code to test that
        # tuples are immutable

        tuple1 = (0, 1, 2, 3)
        tuple1[0] = 4
        print(tuple1)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[3], line 5
      1 # Python code to test that
      2 # tuples are immutable
      4 tuple1 = (0, 1, 2, 3)
----> 5 tuple1[0] = 4
      6 print(tuple1)

TypeError: 'tuple' object does not support item assignment
```

```
In [5]: # Python code to test that
        # strings are immutable

        message = "Welcome to GeeksforGeeks"
        message[0] = 'p'
        print(message)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[5], line 5
      1 # Python code to test that
      2 # strings are immutable
      4 message = "Welcome to GeeksforGeeks"
----> 5 message[0] = 'p'
      6 print(message)

TypeError: 'str' object does not support item assignment
```

```
In [ ]:
```

Challenge

1. Create a list of your favorite movies and print it.
2. Add a new movie to the list and print the updated list.
3. Create a tuple of your favorite books and print it.
4. Try to change an item in the tuple and see what happens.
5. Write a program that prints each fruit in the list on a new line.

---

7. Dictionaries

Dictionaries are used to store data values in key:value pairs.

Example Code

Dictionary

Challenge

1. Create a dictionary to store your favorite book's title, author, and year published.

2. Print the author's name from the dictionary.
3. Add a new key-value pair for the genre of the book.
4. Write a program that prints all keys in the dictionary.
5. Write a program that prints all values in the dictionary.

In [5]:
```
person = {
    "name": "Alice",
    "age": 25,
    "city": "New York"
}

print(person["name"])
```

```
Alice
```

Examples for reference

These problems will help you practice Python programming concepts such as `if-else`, `for loop`, `while loop`, `lists`, `tuples`, `dictionaries`, etc. Each problem is related to real-world scientific fields like physics, mathematics, chemistry, and geology.

---

1. Simple Temperature Converter (Chemistry)

- **Concepts**: `if-else`
- Write a Python program that converts temperature between Celsius, Fahrenheit, and Kelvin, based on user input.
- **Example**: Input the temperature in Celsius and convert it to Fahrenheit and Kelvin.

---

2. Gravitational Force Calculator (Physics)

- **Concepts**: `input`, `for loop`
- Create a program that calculates the gravitational force between two masses using Newton's law of gravitation.
- Formula: $F = \frac{G \cdot m_1 \cdot m_2}{r^2}$
- **Example**: Input two masses and the distance between them.

---

3. Prime Number Checker (Mathematics)

- **Concepts**: `if-else`, `for loop`
- Write a program that checks if a number is prime.
- **Example**: Input a number and the program should display whether it is prime or not.

---

4. Fibonacci Sequence Generator (Mathematics)

- **Concepts**: `while loop`, `lists`
- Write a Python program to generate the first `n` numbers in the Fibonacci sequence.
- **Example**: Input the number of terms, and output the sequence.

---

5. pH Value Classifier (Chemistry)

- **Concepts**: `if-else`
- Create a program that takes the pH value of a solution as input and classifies it as acidic, neutral, or basic.
- **Example**: Input a pH value and the program classifies it as acidic (<7), neutral (=7), or basic (>7).

---

6. Rock Classification (Geology)

- **Concepts**: `if-else`, `lists`, `tuples`
- Create a program that classifies rocks based on their types (igneous, sedimentary, metamorphic). Store examples of rocks in lists/tuples for each type.
- **Example**: Input a rock name, and the program outputs its classification.

---

7. Projectile Motion Calculator (Physics)

- **Concepts**: `while loop`, `math library`
- Write a program that calculates the range and maximum height of a projectile given initial velocity and launch angle.
- Use the equations:
  - $R = \frac{v^2 \sin(2\theta)}{g}$
  - $H = \frac{v^2 \sin^2(\theta)}{2g}$
- **Example**: Input initial velocity and angle, output range and height.

---

8. Matrix Multiplication (Mathematics)

- **Concepts**: `nested loops`, `lists`
- Write a Python program to perform matrix multiplication.
- **Example**: Input two matrices and output their product.

---

9. Periodic Table Lookup (Chemistry)

- **Concepts**: `dictionaries`
- Create a program that stores atomic numbers and symbols of the first 20 elements of the periodic table in a dictionary. Allow the user to look up an element by its atomic number or symbol.
- **Example**: Input an atomic number or symbol, and output the element's name.

---

10. Earthquake Magnitude Classifier (Geology)

- **Concepts**: `if-else`
- Write a program that classifies earthquakes based on the Richter scale magnitude.
- **Example**: Input a magnitude and classify the earthquake (e.g., Minor: <4.0, Moderate: 5.0-5.9, Major: ≥7.0).

---

11. Factorial Calculator (Mathematics)

- **Concepts**: `for loop`

- Write a program that calculates the factorial of a given number.
- **Example**: Input a number, and the program returns its factorial.

---

12. Simulating Radioactive Decay (Physics)

- **Concepts**: `while loop`
- Write a program that simulates radioactive decay using a simple decay law: $N(t) = N_0 e^{-\lambda t}$
- **Example**: Input initial number of atoms, decay constant, and time, then output the remaining number of atoms.

---

13. Distance Between Two Points (Mathematics)

- **Concepts**: `tuples`, `math library`
- Write a program that calculates the distance between two points in a 2D plane.
- Use the distance formula: $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$
- **Example**: Input two points as tuples, and output the distance.

---

14. Planetary Weight Calculator (Physics)

- **Concepts**: `if-else`, `lists`
- Create a program that calculates a person's weight on different planets using their weight on Earth and the gravitational factor for each planet.
- **Example**: Input weight on Earth, choose a planet, and the program returns the weight on that planet.

---

15. Unit Converter (General)

- **Concepts**: `if-else`, `dictionaries`
- Write a program that converts units (e.g., kilometers to miles, grams to ounces, Celsius to Fahrenheit).
- **Example**: Input a value and its unit, and the program converts it to the desired unit.

In [ ]:
```

```

---

1. Simple Temperature Converter (Chemistry)

Question:

Write a Python program that converts temperature from Celsius to both Fahrenheit and Kelvin. Use the following formulas:

- Fahrenheit = $\text{Celsius} \times \frac{9}{5} + 32$
- Kelvin = $\text{Celsius} + 273.15$

Answer:

```python
def temperature_converter(celsius):
    fahrenheit = (celsius * 9/5) + 32
    kelvin = celsius + 273.15
    return fahrenheit, kelvin

celsius = float(input("Enter temperature in Celsius: "))
fahrenheit, kelvin = temperature_converter(celsius)
print(f"Temperature in Fahrenheit: {fahrenheit}")
print(f"Temperature in Kelvin: {kelvin}")
```

---

2. Gravitational Force Calculator (Physics)

Question:

Create a program that calculates the gravitational force between two masses using the formula: $F = \frac{G \cdot m_1 \cdot m_2}{r^2}$ Where $G = 6.674 \times 10^{-11}$ is the gravitational constant.

Answer:

```python
def gravitational_force(m1, m2, r):
    G = 6.674 * 10**-11
    force = (G * m1 * m2) / r**2
    return force

m1 = float(input("Enter the mass of the first object (kg): "))
m2 = float(input("Enter the mass of the second object (kg): "))
r = float(input("Enter the distance between the objects (m): "))

force = gravitational_force(m1, m2, r)
print(f"The gravitational force is: {force} N")
```

---

3. Prime Number Checker (Mathematics)

Question:

Write a program to check if a number is prime.

Answer:

```python
def is_prime(number):
    if number <= 1:
        return False
    for i in range(2, int(number**0.5) + 1):
        if number % i == 0:
            return False
    return True

num = int(input("Enter a number: "))
if is_prime(num):
    print(f"{num} is a prime number.")
else:
    print(f"{num} is not a prime number.")
```

---

4. Fibonacci Sequence Generator (Mathematics)

Question:

Write a program to generate the first `n` numbers in the Fibonacci sequence.

Answer:

```python
def fibonacci(n):
    sequence = [0, 1]
    while len(sequence) < n:
        sequence.append(sequence[-1] + sequence[-2])
    return sequence[:n]

n = int(input("Enter the number of Fibonacci terms to generate: "))
fib_sequence = fibonacci(n)
print(f"The first {n} terms of the Fibonacci sequence: {fib_sequence}")
```

---

5. pH Value Classifier (Chemistry)

Question:

Write a program that classifies a solution as acidic, neutral, or basic based on its pH value.

Answer:

```python
def classify_ph(ph_value):
    if ph_value < 7:
        return "Acidic"
    elif ph_value == 7:
        return "Neutral"
    else:
        return "Basic"

ph_value = float(input("Enter the pH value: "))
classification = classify_ph(ph_value)
print(f"The solution is: {classification}")
```

---

6. Rock Classification (Geology)

Question:

Create a program that classifies rocks as igneous, sedimentary, or metamorphic. The program should store example rock names in tuples for each type.

Answer:

```python
def classify_rock(rock):
    igneous = ("Granite", "Basalt", "Obsidian")
    sedimentary = ("Limestone", "Shale", "Sandstone")
    metamorphic = ("Marble", "Slate", "Schist")

    if rock in igneous:
        return "Igneous"
    elif rock in sedimentary:
        return "Sedimentary"
    elif rock in metamorphic:
        return "Metamorphic"
    else:
        return "Unknown"

rock_name = input("Enter the name of the rock: ").capitalize()
classification = classify_rock(rock_name)
print(f"{rock_name} is a {classification} rock.")
```

---

7. Projectile Motion Calculator (Physics)

Question:

Write a program that calculates the range and maximum height of a projectile given initial velocity and launch angle.

Answer:

```python
import math

def projectile_motion(velocity, angle):
    g = 9.81  # acceleration due to gravity (m/s^2)
    angle_rad = math.radians(angle)
    range_proj = (velocity**2 * math.sin(2*angle_rad)) / g
    max_height = (velocity**2 * math.sin(angle_rad)**2) / (2*g)
    return range_proj, max_height

velocity = float(input("Enter the initial velocity (m/s): "))
angle = float(input("Enter the launch angle (degrees): "))

range_proj, max_height = projectile_motion(velocity, angle)
print(f"Range: {range_proj:.2f} meters")
print(f"Maximum Height: {max_height:.2f} meters")
```

---

8. Matrix Multiplication (Mathematics)

Question:

Write a program to perform matrix multiplication between two matrices.

Answer:

```python
def matrix_multiply(A, B):
    result = [[0 for _ in range(len(B[0]))] for _ in range(len(A))]
    for i in range(len(A)):
        for j in range(len(B[0])):
            for k in range(len(B)):
                result[i][j] += A[i][k] * B[k][j]
    return result

A = [[1, 2], [3, 4]]
B = [[5, 6], [7, 8]]

result = matrix_multiply(A, B)
print("Result of matrix multiplication:")
for row in result:
    print(row)
```

---

9. Periodic Table Lookup (Chemistry)

Question:

Create a program that stores the atomic numbers and symbols of the first 20 elements in a dictionary and allows the user to look up elements by atomic number or symbol.

Answer:

```python
periodic_table = {
    1: "H", 2: "He", 3: "Li", 4: "Be", 5: "B", 6: "C", 7: "N", 8: "O", 9: "F", 10: "Ne",
    11: "Na", 12: "Mg", 13: "Al", 14: "Si", 15: "P", 16: "S", 17: "Cl", 18: "Ar", 19: "K", 20: "Ca"
}

def lookup_element(query):
    if isinstance(query, int):
        return periodic_table.get(query, "Unknown element")
    elif isinstance(query, str):
        for number, symbol in periodic_table.items():
            if symbol == query.capitalize():
                return number
    return "Unknown element"

query = input("Enter atomic number or symbol: ")
if query.isdigit():
    result = lookup_element(int(query))
else:
    result = lookup_element(query)

print(f"Result: {result}")
```

---

10. Earthquake Magnitude Classifier (Geology)

Question:

Write a program that classifies earthquakes based on the Richter scale magnitude.

Answer:

```python
def classify_earthquake(magnitude):
    if magnitude < 4.0:
        return "Minor"
    elif 4.0 <= magnitude < 5.0:
        return "Light"
    elif 5.0 <= magnitude < 6.0:
        return "Moderate"
    elif 6.0 <= magnitude < 7.0:
        return "Strong"
    else:
        return "Major"

magnitude = float(input("Enter the earthquake magnitude: "))
classification = classify_earthquake(magnitude)
print(f"Earthquake classification: {classification}")
```

```
In [ ]:
```

Assignment 3

```
1. There are a total of 5 problems. Each question is compulsory.
2. Each question carries 2 marks.
3. Provide neat and clean code with comments and best practices.
```

11. Factorial Calculator (Mathematics)

Question:

Write a Python program that calculates the factorial of a given number `n`. The factorial of a number `n` is defined as: $n! = n \times (n-1) \times (n-2) \times \cdots \times 1$

Hint:

- You can use a `for loop` to multiply the numbers from 1 to `n`.
- Consider initializing a variable, `fact`, to 1, and then multiply it by each number in the range from 1 to `n`.

---

12. Simulating Radioactive Decay (Physics)

Question:

Write a program that simulates radioactive decay using the decay formula: $N(t) = N_0 e^{-\lambda t}$ Where:

- N_0 is the initial number of atoms,
- λ is the decay constant,
- t is time.

Hint:

- Use the `math.exp()` function to compute the exponential decay.
- You'll need to input the values for N_0, λ, and t.
- The decay formula gives the number of atoms left after a certain time, so your program should output that number.

---

13. Distance Between Two Points (Mathematics)

Question:

Write a program that calculates the distance between two points in a 2D plane. The distance between points (x_1, y_1) and (x_2, y_2) is given by: $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

Hint:

- Use the `math.sqrt()` function to calculate the square root.
- You'll need to ask the user for the coordinates of the two points and then apply the distance formula.
- Make sure you use `**2` to calculate squares in Python.

---

14. Planetary Weight Calculator (Physics)

Question:

Write a program that calculates a person's weight on different planets in the solar system. Given their weight on Earth, calculate their weight on another planet using the planet's gravitational factor relative to Earth's gravity.

Hint:

- Create a dictionary where the keys are planet names and the values are their gravitational factors (relative to Earth).

- The formula is:
  $\text{weight on planet} = \text{weight on Earth} \times \text{gravitational factor}$
- Ask the user for their weight on Earth and the planet they want to know their weight on.

---

15. Unit Converter (General)

Question:

Write a Python program that converts units such as kilometers to miles, grams to ounces, or Celsius to Fahrenheit. Let the user choose the conversion type and input a value to be converted.

Hint:

- Use a dictionary to store conversion factors between units.
- Based on the user's input, choose the correct conversion factor and apply it to the value.
- You can use multiple `if-else` statements to check which conversion the user wants to perform.

```
In [ ]:
```

```
In [ ]:
```

Understanding Modules in Python

What is a Module?

A **module** in Python is simply a file containing Python code. It can define functions, classes, and variables, and can also include runnable code. Modules allow you to break down large programs into smaller, more manageable pieces of code.

Why Use Modules?

- **Code Reusability**: Write code once and reuse it across different programs.
- **Organization**: Group related functions, classes, or variables together in a file.
- **Maintainability**: Easier to debug and update smaller sections of code.
- **Namespace Management**: Avoid name clashes by separating code into different modules.

How to Create a Module

Any Python file (.py) can be treated as a module. For example, you can create a file called `my_module.py`:

```python
# my_module.py

def greet(name):
    return f"Hello, {name}!"

PI = 3.14159
```

Importing a Module

You can import a module into another script using the `import` statement. This allows you to access the functions and variables defined in that module.

```
# Importing my_module in another script

import my_module

print(my_module.greet("Class"))  # Output: Hello, Class!
print(my_module.PI)              # Output: 3.14159
```

```
In [ ]:
```

```
In [79]: import module


print(module.greet("Class"))  # Output: Hello, Class!
print(module.PI)              # Output: 3.14159
```

```
Hello, Class!
3.14159
```

```
In [ ]:
```

Importing Specific Functions or Variables

You can also import specific functions or variables from a module:

```
from my_module import greet, PI

print(greet("Class"))  # Output: Hello, Class!
print(PI)              # Output: 3.14159
```

```
In [ ]:
```

```
In [ ]:
```

Renaming Modules

You can give an imported module a different name using the `as` keyword:

```
import my_module as mm

print(mm.greet("Class"))  # Output: Hello, Class!
```

```
In [ ]:
```

Built-in Python Modules

Python has a large standard library of built-in modules you can use, such as `math`, `random`, `os`, etc.

Example of using the `math` module:

```
import math

print(math.sqrt(16))   # Output: 4.0
print(math.pi)         # Output: 3.141592653589793
```

Exploring Module Content

You can use the `dir()` function to list all the functions and variables defined in a module:

```
import math
print(dir(math))
```

Conclusion

Modules are essential for writing clean, organized, and reusable code. By breaking down complex programs into smaller modules, you can make your code more manageable and maintainable.

```
In [ ]:
```

Here's a guide on the basic usage of the NumPy module along with some examples of routines that are useful for scientific purposes:

Introduction to NumPy

NumPy is a powerful library in Python that is used for numerical computing. It allows the creation and manipulation of large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays.

Installation

If NumPy is not already installed, you can install it using pip:

```
sudo apt-get install python3-pip
pip install numpy
```

Basic Usage of NumPy

1. **Creating Arrays**

```
import numpy as np

# Creating a 1D array
arr1 = np.array([1, 2, 3])

# Creating a 2D array
arr2 = np.array([[1, 2, 3], [4, 5, 6]])

# Creating arrays with zeros and ones
zeros = np.zeros((2, 3))  # 2x3 array of zeros
ones = np.ones((2, 3))    # 2x3 array of ones

# Creating arrays with a range of values
arr_range = np.arange(0, 10, 2)  # From 0 to 9, step 2

# Creating arrays with equally spaced numbers
lin_space = np.linspace(0, 1, 5)  # 5 numbers between 0 and 1
```

2. **Array Attributes**

```
# Getting the shape of an array
print(arr2.shape)  # Output: (2, 3)

# Getting the number of dimensions
```

```python
print(arr2.ndim)  # Output: 2

# Getting the size (number of elements)
print(arr2.size)  # Output: 6

# Getting the data type of elements
print(arr2.dtype)  # Output: int64 (or similar)
```

3. **Reshaping Arrays**

```python
# Reshaping a 1D array into a 2D array
reshaped = arr1.reshape(1, 3)  # Output: array([[1, 2, 3]])
```

4. **Mathematical Operations**

```python
# Element-wise operations
arr3 = np.array([1, 2, 3])
arr4 = np.array([4, 5, 6])

sum_arr = arr3 + arr4  # Output: array([5, 7, 9])
prod_arr = arr3 * arr4  # Output: array([ 4, 10, 18])

# Scalar operations
arr_scalar = arr3 * 2  # Output: array([2, 4, 6])

# Universal functions (ufuncs)
sin_arr = np.sin(arr3)  # Applies sine function element-wise
```

Useful NumPy Routines for Scientific Purposes

1. **Random Number Generation**

```python
# Generating random numbers
rand_nums = np.random.random((3, 3))  # 3x3 array of random numbers between
0 and 1

# Normal distribution
normal_dist = np.random.normal(0, 1, (3, 3))  # Mean 0, standard deviation 1
```

2. **Statistical Functions**

```python
data = np.array([1, 2, 3, 4, 5])

# Mean
mean = np.mean(data)  # Output: 3.0

# Standard deviation
std_dev = np.std(data)  # Output: 1.414...

# Median
median = np.median(data)  # Output: 3.0
```

3. **Linear Algebra**

```python
# Matrix multiplication
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])

matmul = np.dot(A, B)  # Output: [[19, 22], [43, 50]]
```

```
# Matrix inverse
inv_A = np.linalg.inv(A)  # Output: [[-2. ,  1. ], [ 1.5, -0.5]]

# Eigenvalues and eigenvectors
eigen_vals, eigen_vecs = np.linalg.eig(A)
```

4. **Integration with Other Libraries**

- **SciPy**: NumPy arrays are compatible with the SciPy library, which offers more advanced mathematical operations (integration, optimization, etc.).
- **Pandas**: NumPy arrays are the foundation of the Pandas library, which is widely used for data manipulation and analysis.

5. **Fast Fourier Transform (FFT)**

```
# Computing the Fast Fourier Transform
x = np.array([0, 1, 2, 3])
fft_x = np.fft.fft(x)  # Output: [ 6.+0.j -2.+2.j -2.+0.j -2.-2.j]
```

6. **Saving and Loading Arrays**

```
# Saving an array to a file
np.save('my_array.npy', arr1)

# Loading an array from a file
loaded_arr = np.load('my_array.npy')
```

Conclusion

NumPy is a fundamental library for scientific computing in Python, providing powerful array operations and mathematical functions. Its integration with libraries like SciPy and Pandas makes it an essential tool for data analysis, machine learning, and scientific research.

In [ ]:
```
import numpy as np


a = np.array([[1,2], [3,5]])
```

Problems for practice

Problem 1: Creating Arrays

Write a Python program that creates a 3x3 matrix with values ranging from 1 to 9, and then print the matrix.

Problem 2: Array Slicing

Given the following 3D NumPy array:

```
arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
```
Write a program to extract the second column from the array.

Problem 3: Arithmetic Operations

Create two NumPy arrays:

- `arr1 = np.array([1, 2, 3, 4, 5])`
- `arr2 = np.array([5, 4, 3, 2, 1])`

Write a program to perform the following operations:

- Add the two arrays.
- Multiply the two arrays.
- Find the element-wise difference between `arr1` and `arr2`.

Problem 4: Broadcasting

Write a program to add a constant value of 10 to each element of the following array:

```
arr = np.array([[10, 20, 30], [40, 50, 60]])
```

Problem 5: Reshaping Arrays

Given the following array:

```
arr = np.arange(12)

##### Answer
new_arr = arr.reshape(4, 3)
print(new_arr)
```

Write a program to reshape this 1D array into a 3x4 2D array.

Problem 6: Statistical Functions

Given the following array of exam scores:

```
scores = np.array([55, 89, 76, 65, 93, 42, 67])
```

Write a program to calculate and print the mean, median, and standard deviation of the scores.

Problem 7: Random Numbers

Write a program to generate a 4x4 matrix of random numbers sampled from a normal distribution with a mean of 0 and a standard deviation of 1.

Problem 8: Matrix Multiplication

Given the following two matrices:

```
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])
```

Write a program to compute the dot product of matrices `A` and `B`.

Problem 9: Boolean Indexing

Given the following array of numbers:

```
arr = np.array([10, 25, 33, 45, 55, 67, 72, 89, 91])
```

Write a program to extract all numbers from the array that are greater than 50.

Problem 10: Element-wise Conditional

Write a program to replace all elements in the following array that are greater than 5 with 0, and leave the other elements unchanged:

```
arr = np.array([1, 4, 6, 8, 10])
```

In [ ]:

In [ ]:

A beginners guide on the usage of the **Pandas** module, along with examples that are useful for data manipulation and analysis in scientific and practical contexts.

Introduction to Pandas

Pandas is an open-source library that provides high-performance, easy-to-use data structures and data analysis tools for Python. It is built on top of NumPy and is widely used for data manipulation, cleaning, and analysis tasks.

Installation

If Pandas is not already installed, you can install it using pip:

```
pip install pandas
```

Basic Usage of Pandas

1. **Creating DataFrames**

A DataFrame is a 2-dimensional, size-mutable, and potentially heterogeneous data structure. It is similar to an Excel spreadsheet or SQL table.

```
import pandas as pd

# Creating a DataFrame from a dictionary
data = {
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Age': [25, 30, 35],
    'Score': [85, 90, 95]
}
df = pd.DataFrame(data)

# Displaying the DataFrame
print(df)
```

2. **Loading Data from Files**

Pandas allows you to read and write data from various file formats such as CSV, Excel, JSON, and SQL databases.

```
# Reading a CSV file into a DataFrame
df = pd.read_csv('data.csv')

# Writing a DataFrame to a CSV file
df.to_csv('output.csv', index=False)
```

3. **DataFrame Attributes**

```
# Getting basic information about the DataFrame
print(df.shape)  # Number of rows and columns
```

```python
print(df.columns)  # Column names
print(df.dtypes)  # Data types of each column

# Quick statistics summary of numerical data
print(df.describe())
```

4. **Selecting Data**

```python
# Selecting a single column
ages = df['Age']

# Selecting multiple columns
subset = df[['Name', 'Score']]

# Selecting rows by index
row_0 = df.iloc[0]  # First row
rows_1_to_3 = df.iloc[1:3]  # Second and third rows

# Selecting rows by condition
adults = df[df['Age'] > 18]
```

Useful Pandas Routines for Scientific Purposes

1. **Handling Missing Data**

```python
# Filling missing data with a specified value
df.fillna(0, inplace=True)

# Dropping rows with missing values
df.dropna(inplace=True)

# Checking for missing values
print(df.isnull().sum())
```

2. **Data Manipulation**

```python
# Adding a new column
df['Passed'] = df['Score'] > 50

# Renaming columns
df.rename(columns={'Name': 'Student Name', 'Score': 'Exam Score'},
inplace=True)

# Dropping columns
df.drop(columns=['Age'], inplace=True)
```

3. **Sorting and Ranking**

```python
# Sorting by column values
df_sorted = df.sort_values(by='Score', ascending=False)

# Ranking the data
df['Rank'] = df['Score'].rank(ascending=False)
```

4. **Grouping and Aggregation**

```python
# Grouping by a column and applying aggregate functions
grouped = df.groupby('Passed').agg({
    'Score': ['mean', 'min', 'max'],
    'Age': 'mean'
})
```

```python
# Resetting the index
grouped.reset_index(inplace=True)
```

5. **Merging and Joining DataFrames**

```python
# Creating another DataFrame
df2 = pd.DataFrame({
    'Name': ['Alice', 'Charlie', 'David'],
    'City': ['New York', 'Los Angeles', 'Chicago']
})

# Merging DataFrames on a common column
merged_df = pd.merge(df, df2, on='Name', how='left')  # Left join
```

6. **Time Series Data**

Pandas is great for handling time series data, with built-in support for date and time data types.

```python
# Creating a time series
dates = pd.date_range('20230101', periods=6)
df_time = pd.DataFrame({'Date': dates, 'Value': [10, 15, 20, 25, 30, 35]})

# Setting the index to the Date column
df_time.set_index('Date', inplace=True)

# Resampling the data (e.g., converting daily data to monthly data)
monthly_data = df_time.resample('M').mean()
```

7. **Statistical Functions**

```python
# Applying statistical functions
mean_score = df['Score'].mean()  # Mean of the 'Score' column
median_age = df['Age'].median()  # Median of the 'Age' column

# Cumulative sum
df['Cumulative Score'] = df['Score'].cumsum()

# Correlation between numerical columns
correlation = df.corr()
```

8. **Visualization with Pandas**

Pandas has built-in support for basic plotting using Matplotlib. You can visualize your data with just one line of code.

```python
import matplotlib.pyplot as plt

# Plotting a column
df['Score'].plot(kind='line')
plt.show()

# Plotting histograms
df['Score'].plot(kind='hist')
plt.show()
```

Conclusion

Pandas is a powerful and flexible library for data manipulation, offering DataFrames and Series for managing structured data. Whether you're working with CSV files, time series data, or performing complex group operations, Pandas simplifies the workflow for data analysis. It integrates seamlessly with libraries like NumPy, Matplotlib, and Scikit-learn, making it a versatile tool in scientific computing.

```
In [ ]:
```

Problem 1: Creating a DataFrame

Create a DataFrame from the following dictionary:

```
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eva'],
    'Age': [25, 30, 35, 40, 45],
    'Score': [85, 90, 95, 100, 88]
}
```

Write a program to create the DataFrame and print it.

Problem 2: Selecting Columns

Given the following DataFrame:

```
df = pd.DataFrame({
    'Name': ['Tom', 'Jerry', 'Spike', 'Tyke'],
    'Age': [22, 21, 23, 20],
    'Grade': [88, 92, 85, 95]
})
```

Write a program to select and print only the `Name` and `Grade` columns.

Problem 3: Filtering Rows

Using the DataFrame from **Problem 2**, write a program to filter out and display only the rows where the `Grade` is greater than 90.

Problem 4: Adding a New Column

Using the DataFrame from **Problem 2**, write a program to add a new column named `Pass` that contains `True` if the `Grade` is greater than 85 and `False` otherwise.

Problem 5: Reading Data from CSV

Write a program to read a CSV file into a DataFrame. Assume the CSV file is named `students.csv` and contains the columns `Name`, `Age`, and `Score`. Print the first 5 rows of the DataFrame.

Problem 6: Handling Missing Data

Write a program to create the following DataFrame:

```
data = {
    'Name': ['Anna', 'Brian', 'Cathy', 'Diana'],
    'Age': [23, None, 35, 29],
    'Score': [85, 88, None, 90]
}
df = pd.DataFrame(data)
```

- Replace all missing values in the `Age` column with the mean age.
- Replace all missing values in the `Score` column with 0.

- Print the updated DataFrame.

Problem 7: Grouping and Aggregation

Given the following DataFrame:

```python
df = pd.DataFrame({
    'Team': ['A', 'B', 'A', 'B', 'A', 'B'],
    'Player': ['Alice', 'Bob', 'Charlie', 'David', 'Eva', 'Frank'],
    'Score': [88, 95, 82, 91, 89, 87]
})
```

Write a program to group the data by `Team` and calculate the average `Score` for each team.

Problem 8: Sorting Data

Given the following DataFrame:

```python
df = pd.DataFrame({
    'Name': ['Xander', 'Yvonne', 'Zara', 'Walter'],
    'Age': [35, 28, 40, 33],
    'Salary': [70000, 65000, 80000, 72000]
})
```

Write a program to sort the DataFrame by `Salary` in descending order and print the result.

Problem 9: Merging DataFrames

Given the following two DataFrames:

```python
df1 = pd.DataFrame({
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Score': [85, 90, 95]
})

df2 = pd.DataFrame({
    'Name': ['Alice', 'Charlie', 'David'],
    'City': ['New York', 'Los Angeles', 'Chicago']
})
```

Write a program to merge these two DataFrames on the `Name` column and display the result. Use a left join.

Problem 10: Working with Time Series

Write a program to:

- Create a time series of daily temperatures for 7 days starting from '2023-01-01'.
- Create a DataFrame that contains a `Date` column with the dates and a `Temperature` column with the following values: `[32, 35, 28, 30, 31, 29, 33]`.
- Set the `Date` column as the index of the DataFrame and print the result.

These problems test the core concepts of **Pandas**, such as DataFrame creation, column and row selection, filtering, handling missing data, grouping, sorting, merging, and working with time series.

```
In [ ]:
```