# Predefined functions, Basic Maths operations, Math library.

Cosmic Charade


In this section, we'll explore Python's predefined functions, how to perform basic mathematical operations, and how to leverage the `math` library for more advanced mathematical computations. We will provide extraordinary problems and interpret their implementations in Python.

## 1. Predefined Functions

Python offers many predefined (built-in) functions, which are ready-to-use. These functions help perform tasks like input/output operations, conversions, and mathematical calculations without needing external libraries.

### 1.1 What are Predefined Functions?

Predefined functions, also known as built-in functions, are functions that are always available in Python without needing to import any additional modules. They perform common tasks and simplify coding by providing reusable functionality.

### 1.2 Common Predefined Functions

Here are some frequently used predefined functions:

- **`print()`**: Outputs data to the console.
- **`len()`**: Returns the length of an object.
- **`type()`**: Returns the type of an object.
- **`int()`**, **`float()`**, **`str()`**: Convert data types.
- **`range()`**: Generates a sequence of numbers.
- **`input()`**: Takes user input.
- **`abs()`**: Returns the absolute value of a number.
- **`min()`**, **`max()`**: Return the smallest and largest item in an iterable.
- **`sum()`**: Sums items of an iterable.

### 1.3 Examples and Implementations

#### Example 1: Using `len()` and `type()`

```
# Using len() to get the length of a list
fruits = ["apple", "banana", "cherry", "date"]
print("Number of fruits:", len(fruits))  # Output: 4

# Using type() to check data types
number = 10
pi = 3.1415
name = "Python"
print(type(number))  # Output: <class 'int'>
print(type(pi))      # Output: <class 'float'>
print(type(name))    # Output: <class 'str'>
```

```
Number of fruits: 4
<class 'int'>
<class 'float'>
<class 'str'>
```

**Interpretation:**

- `len(fruits)` returns `4` because there are four items in the `fruits` list.
- `type()` function confirms the data types of different variables.

### Example 2: Type Conversion with `int()`, `float()`, `str()`

```
# Type conversion examples
num_str = "25"
num_int = int(num_str)  # Converts string to integer
num_float = float(num_int)  # Converts integer to float
num_str_again = str(num_float)  # Converts float back to string

print(num_int, type(num_int))          # Output: 25 <class 'int'>
print(num_float, type(num_float))      # Output: 25.0 <class 'float'>
print(num_str_again, type(num_str_again))  # Output: '25.0' <class 'str'>
```

```
25 <class 'int'>
25.0 <class 'float'>
25.0 <class 'str'>
```

**Interpretation:**

- The string `"25"` is converted to an integer `25` using `int()`.
- The integer `25` is converted to a float `25.0` using `float()`.
- The float `25.0` is converted back to a string `"25.0"` using `str()`.

### Example 3: Using `range()` in Loops

```
# Using range() to generate numbers from 0 to 4
for i in range(5):
    print(i)
```

```
0
1
2
3
4
```

**Interpretation:**

- `range(5)` generates numbers from `0` to `4`. The `for` loop iterates over these numbers, printing each one.

## 2. Basic Math Operations in Python

Python supports a wide range of mathematical operations, from basic arithmetic to more complex computations.

### 2.1 Arithmetic Operators

- **Addition (+)**
- **Subtraction (-)**
- **Multiplication (*)**
- **Division (/)**
- **Floor Division (//)**
- **Modulus (%)**
- **Exponentiation (**)**

### 2.2 Examples and Implementations

#### Example 1: Basic Arithmetic Operations

```
a = 15
b = 4

# Addition
add = a + b
print(f"{a} + {b} = {add}")  # Output: 15 + 4 = 19

# Subtraction
sub = a - b
print(f"{a} - {b} = {sub}")  # Output: 15 - 4 = 11

# Multiplication
mul = a * b
print(f"{a} * {b} = {mul}")  # Output: 15 * 4 = 60

# Division
div = a / b
print(f"{a} / {b} = {div}")  # Output: 15 / 4 = 3.75
```

```python
# Floor Division
floor_div = a // b
print(f"{a} // {b} = {floor_div}")  # Output: 15 // 4 = 3

# Modulus
mod = a % b
print(f"{a} % {b} = {mod}")  # Output: 15 % 4 = 3

# Exponentiation
exp = a ** b
print(f"{a} ** {b} = {exp}")  # Output: 15 ** 4 = 50625
```

```
15 + 4 = 19
15 - 4 = 11
15 * 4 = 60
15 / 4 = 3.75
15 // 4 = 3
15 % 4 = 3
15 ** 4 = 50625
```

**Interpretation:**

- Basic arithmetic operations are straightforward. The floor division `//` returns the integer part of the division, and modulus `%` returns the remainder.
- Exponentiation `**` raises `a` to the power of `b`.

### Example 2: Order of Operations

```python
# Demonstrating order of operations
result = 3 + 4 * 2 / (1 - 5) ** 2
print("Result:", result)  # Output: Result: 3.5
```

```
Result: 3.5
```

**Interpretation:**

- Python follows the standard mathematical order of operations (PEMDAS/BODMAS).
- Calculation steps:
  1. Parentheses: `(1 - 5)` = `-4`

2. Exponentiation: `(-4) ** 2` = `16`
3. Multiplication and Division: `4 * 2 / 16` = `8 / 16` = `0.5`
4. Addition: `3 + 0.5` = `3.5`

**Example 3: Using Augmented Assignment Operators**

```
# Using augmented assignment operators
count = 10
count += 5  # Equivalent to count = count + 5
print(count)  # Output: 15

count *= 2  # Equivalent to count = count * 2
print(count)  # Output: 30

count -= 10  # Equivalent to count = count - 10
print(count)  # Output: 20
```

```
15
30
20
```

**Interpretation:**

- Augmented assignment operators provide a shorthand for updating the value of a variable based on its current value.

### 3. The Math Library in Python

The `math` module provides access to mathematical functions defined by the C standard. It offers a wide range of functions for performing mathematical operations beyond basic arithmetic.

#### 3.1 Importing the Math Module

Before using the functions in the `math` module, you need to import it:

```
import math
```

### 3.2 Common Functions in the Math Module

- **`math.sqrt(x)`**: Returns the square root of `x`.
- **`math.factorial(x)`**: Returns the factorial of `x`.
- **`math.gcd(a, b)`**: Returns the greatest common divisor of `a` and `b`.
- **`math.pi` and `math.e`**: Mathematical constants  and e.
- **`math.sin(x)`, `math.cos(x)`, `math.tan(x)`**: Trigonometric functions.
- **`math.log(x, base)`**: Logarithm of `x` with specified base.
- **`math.ceil(x)`**: Returns the smallest integer greater than or equal to `x`.
- **`math.floor(x)`**: Returns the largest integer less than or equal to `x`.

### 3.3 Examples and Implementations

#### Example 1: Square Root and Factorial

```
import math

# Square root
number = 16
sqrt_num = math.sqrt(number)
print(f"Square root of {number} is {sqrt_num}")  # Output: 4.0

# Factorial
fact_num = 5
factorial = math.factorial(fact_num)
print(f"Factorial of {fact_num} is {factorial}")  # Output: 120
```

```
Square root of 16 is 4.0
Factorial of 5 is 120
```

**Interpretation:**

- `math.sqrt(16)` returns `4.0`.
- `math.factorial(5)` returns `120` because $5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$.

### Example 2: Trigonometric Functions

```
import math

angle_degrees = 45
angle_radians = math.radians(angle_degrees)  # Convert degrees to radians

sin_val = math.sin(angle_radians)
cos_val = math.cos(angle_radians)
tan_val = math.tan(angle_radians)

print(f"Sin({angle_degrees}°) = {sin_val}")  # Output: 0.7071067811865475
print(f"Cos({angle_degrees}°) = {cos_val}")  # Output: 0.7071067811865476
print(f"Tan({angle_degrees}°) = {tan_val}")  # Output: 0.9999999999999999
```

```
Sin(45°) = 0.7071067811865476
Cos(45°) = 0.7071067811865476
Tan(45°) = 0.9999999999999999
```

**Interpretation:**

- Trigonometric functions in the `math` module use radians. The `math.radians()` function converts degrees to radians.
- For 45°, sin and cos both approximately equal `0.7071`, and tan is approximately `1`.

### Example 3: Logarithms and Exponents

```
import math

# Natural logarithm (base e)
x = math.e
ln_x = math.log(x)
print(f"Natural log of {x} is {ln_x}")  # Output: 1.0

# Logarithm base 10
log10_x = math.log(x, 10)
print(f"Log base 10 of {x} is {log10_x}")  # Output: 0.4342944819032518

# Exponentiation
exp_val = math.exp(2)  # e^2
print(f"e^2 is {exp_val}")  # Output: 7.38905609893065
```

```
Natural log of 2.718281828459045 is 1.0
Log base 10 of 2.718281828459045 is 0.43429448190325176
e^2 is 7.38905609893065
```

**Interpretation:**

- `math.log(math.e)` returns `1.0` because the natural logarithm of e is 1.
- `math.log(math.e, 10)` calculates the logarithm base 10 of e.
- `math.exp(2)` calculates e raised to the power of 2.

### Example 4: Greatest Common Divisor (GCD)

```python
import math

a = 48
b = 180
gcd = math.gcd(a, b)
print(f"GCD of {a} and {b} is {gcd}")
```

```
GCD of 48 and 180 is 12
```

### Example 5: Least Common Multiple (LCM)

The **Least Common Multiple (LCM)** of two numbers is the smallest positive integer that is divisible by both numbers. One way to calculate the LCM of two numbers in Python is by using the relationship between LCM and GCD (Greatest Common Divisor):

$$LCM(a, b) = \frac{|a \cdot b|}{GCD(a, b)}$$

Python's `math` library provides a function to compute the GCD, and we can use this to derive the LCM.

```python
import math

# Define the numbers
a = 48
b = 180

# Calculate GCD
```

```
gcd = math.gcd(a, b)
#print(f"GCD of {a} and {b} is {gcd}")

# Calculate LCM using the formula: LCM(a, b) = (a * b) // GCD(a, b)
lcm = (a * b) // gcd
print(f"LCM of {a} and {b} is {lcm}")
```

```
LCM of 48 and 180 is 720
```

**Explanation:**

- **GCD Calculation**: `math.gcd(a, b)` finds the greatest common divisor of `a` and `b`.
- **LCM Calculation**: The LCM is computed using the formula $LCM(a, b) = \frac{|a \cdot b|}{GCD(a, b)}$
- **Output**: The code will first print the GCD and then the calculated LCM.

### 4. Extraordinary Problems with Coding Implementations and Interpretations

To enhance your understanding, let's tackle some challenging problems that utilize predefined functions, basic math operations, and the Math library.

#### Problem 1: Calculate the Hypotenuse of a Right Triangle

**Description:** Given the lengths of the two perpendicular sides of a right triangle, calculate the length of the hypotenuse using the Pythagorean theorem.

**Formula:** $c = \sqrt{a^2 + b^2}$

```
import math

def calculate_hypotenuse(a, b):
    return math.sqrt(a**2 + b**2)

# Example usage
side_a = 3
side_b = 4
hypotenuse = calculate_hypotenuse(side_a, side_b)
print(f"The hypotenuse of a  right triangle with sides {side_a} and {side_b} is {hypotenuse}
```

```
The hypotenuse of a  right triangle with sides 3 and 4 is 5.0
```

**Interpretation:**

- For a right triangle with sides 3 and 4, the hypotenuse calculated using the Pythagorean theorem is 5.0.

### Problem 2: Find All Prime Numbers Up to a Given Number

**Description:** Generate a list of all prime numbers up to a user-specified limit using the Sieve of Eratosthenes algorithm.

**Implementation:**

```python
def sieve_of_eratosthenes(limit):
    if limit < 2:
        return []
    sieve = [True] * (limit + 1)
    sieve[0], sieve[1] = False, False
    for num in range(2, int(math.sqrt(limit)) + 1):
        if sieve[num]:
            for multiple in range(num*num, limit + 1, num):
                sieve[multiple] = False
    primes = [num for num, is_prime in enumerate(sieve) if is_prime]
    return primes

# Example usage
limit = 100
primes = sieve_of_eratosthenes(limit)
print(f"Prime numbers up to {limit}: {primes}")
```

```
Prime numbers up to 100: [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61
```

**Interpretation:**

- The function `sieve_of_eratosthenes` efficiently finds all prime numbers up to the specified limit (30 in this case).

### Problem 3: Calculate Compound Interest

**Description:** Calculate the compound interest based on the principal amount, annual interest rate, number of times interest is compounded per year, and the number of years.

**Formula:** $A = P\left(1 + \frac{r}{n}\right)^{nt}$ Where:

- $A$ = the amount of money accumulated after n years, including interest.
- $P$ = the principal amount.
- $r$ = the annual interest rate (decimal).
- $n$ = the number of times that interest is compounded per year.
- $t$ = the time the money is invested for in years.

```
import math

def calculate_compound_interest(P, r, n, t):
    A = P * math.pow((1 + r / n), n * t)
    return A

# Example usage
principal = 1000  # $1,000
annual_rate = 0.05  # 5%
times_compounded = 12  # Monthly
years = 10

amount = calculate_compound_interest(principal, annual_rate,
                                     times_compounded, years)
print(f"After {years} years, the investment will be worth ${amount:.2f}")
```

```
After 10 years, the investment will be worth $1647.01
```

**Interpretation:**

- An investment of $1,000 at an annual interest rate of 5%, compounded monthly, will grow to $1,647.01 after 10 years.

### Problem 4: Solve a Quadratic Equation

**Description:** Given the coefficients $a$, $b$, and $c$ of a quadratic equation $ax^2 + bx + c = 0$, find the roots using the quadratic formula.

**Formula:** $x = \dfrac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

**Implementation:**

```
import math

def solve_quadratic(a, b, c):
    discriminant = b**2 - 4*a*c
    if discriminant > 0:
        root1 = (-b + math.sqrt(discriminant)) / (2*a)
        root2 = (-b - math.sqrt(discriminant)) / (2*a)
        return (root1, root2)
    elif discriminant == 0:
        root = -b / (2*a)
        return (root,)
    else:
        real_part = -b / (2*a)
        imaginary_part = math.sqrt(-discriminant) / (2*a)
        return (complex(real_part, imaginary_part),
                complex(real_part, -imaginary_part))

# Example usage
coeff_a = 1
coeff_b = -3
coeff_c = 2

roots = solve_quadratic(coeff_a, coeff_b, coeff_c)
print(f"The roots of the equation are: {roots}")
# Output: The roots of the equation are: (2.0, 1.0)
```

```
The roots of the equation are: (2.0, 1.0)
```

**Interpretation:**

- For the quadratic equation $x^2 - 3x + 2 = 0$, the roots are 2.0 and 1.0.

### Problem 5: Calculate the Area of a Circle and Sphere

**Description:** Calculate the area of a circle and the volume of a sphere given the radius.

**Formulas:**

- **Area of a Circle:** $A = \pi r^2$
- **Volume of a Sphere:** $V = \frac{4}{3}\pi r^3$

**Implementation:**

```
import math

def area_of_circle(radius):
    return math.pi * radius ** 2

def volume_of_sphere(radius):
    return (4/3) * math.pi * radius ** 3

# Example usage
r = 5
circle_area = area_of_circle(r)
sphere_volume = volume_of_sphere(r)

print(f"Area of a circle with radius {r} is {circle_area:.2f}")


print(f"Volume of a sphere with radius {r} is {sphere_volume:.2f}")
```

```
Area of a circle with radius 5 is 78.54
Volume of a sphere with radius 5 is 523.60
```

**Interpretation:**

- A circle with a radius of 5 units has an area of approximately 78.54 square units.
- A sphere with a radius of 5 units has a volume of approximately 523.60 cubic units.

### Problem 6: Determine if a Number is a Perfect Square

**Description:** Check whether a given number is a perfect square

**Implementation:**

```
import math

def is_perfect_square(n):
    if n < 0:
        return False
    root = math.isqrt(n)  # Available in Python 3.8+
    return root * root == n

# Example usage
numbers = [16, 20, 25, 26, 0, -4]
for num in numbers:
    result = is_perfect_square(num)
    print(f"{num} is a perfect square: {result}")
```

```
16 is a perfect square: True
20 is a perfect square: False
25 is a perfect square: True
26 is a perfect square: False
0 is a perfect square: True
-4 is a perfect square: False
```

### Problem 7: Generate Fibonacci Sequence Using Recursion

**Description:** Generate the first `n` numbers of the Fibonacci sequence using a recursive approach.

**Implementation:**

```
def fibonacci(n):
    if n <= 0:
        return []
    elif n == 1:
        return [0]
    elif n == 2:
        return [0, 1]
    else:
        seq = fibonacci(n-1)
```

```
        seq.append(seq[-1] + seq[-2])
        return seq

# Example usage
num_terms = 10
fib_sequence = fibonacci(num_terms)
print(f"First {num_terms} terms of Fibonacci sequence: {fib_sequence}")
```

```
First 10 terms of Fibonacci sequence: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

**Interpretation:**

- The recursive function `fibonacci` generates the Fibonacci sequence up to the specified number of terms (`10` in this case).
- The sequence starts with `0` and `1`, and each subsequent number is the sum of the previous two.

### Problem 8: Calculate the angle and distance between two points in a 2D space.

Given points $P(x_1, y_1)$ and $Q(x_2, y_2)$, the distance and angle from $P$ to $Q$ are:

- **Distance:** $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$
- **Angle:** $\theta = \arctan\left(\frac{y_2 - y_1}{x_2 - x_1}\right)$

```
import math

def point_distance_angle(x1, y1, x2, y2):
    # Distance calculation
    distance = math.sqrt((x2 - x1)**2 + (y2 - y1)**2)

    # Angle calculation in radians and conversion to degrees
    angle_rad = math.atan2(y2 - y1, x2 - x1)
    angle_deg = math.degrees(angle_rad)

    return distance, angle_deg

x1, y1 = 0, 0  # Point P
x2, y2 = 3, 4  # Point Q
distance, angle = point_distance_angle(x1, y1, x2, y2)

print(f"Distance: {distance}, Angle: {angle} degrees")
```

```
Distance: 5.0, Angle: 53.13010235415598 degrees
```

**Interpretation:**

- The `math.sqrt()` function computes the distance between two points.
- `math.atan2()` gives the angle between the line connecting the two points and the x-axis. The angle is converted from radians to degrees using `math.degrees()`.

### Curve Plotting in Python

To plot curves in Python, we typically use the `matplotlib` library. This powerful library allows for a variety of 2D plotting capabilities. You can plot simple functions, parametric curves, and even more complex mathematical visualizations.

Here are a few programs that demonstrate different types of curve plotting.

### 1. Plotting a Simple Line: Linear Function

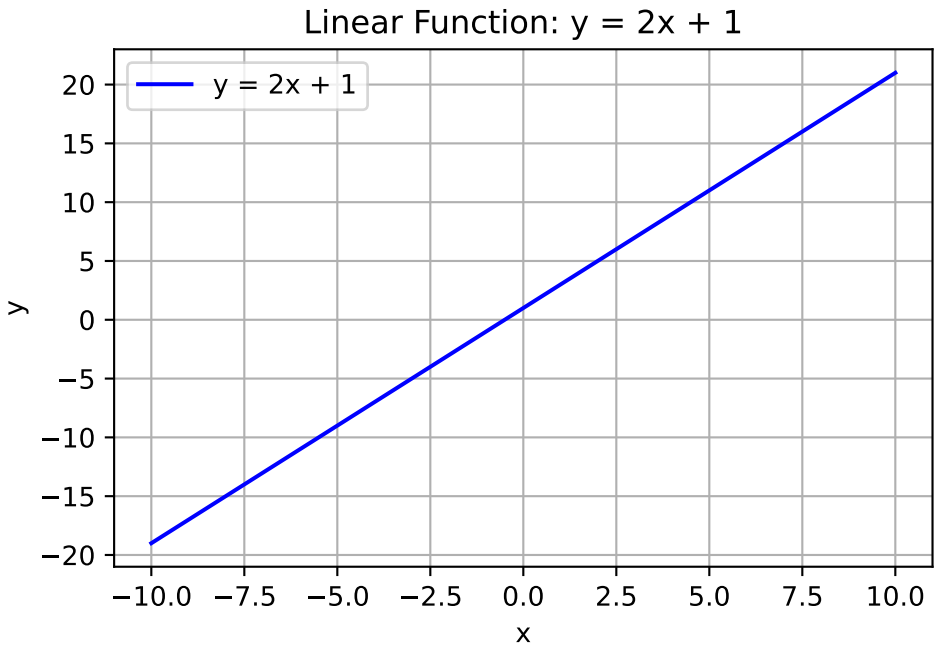
We will plot a simple linear function:

$$y = mx + c$$

**Example: Plot** $y = 2x + 1$

```python
import matplotlib.pyplot as plt
import numpy as np

# Data for the plot
x = np.linspace(-10, 10, 100)  # Generate 100 values from -10 to 10
y = 2 * x + 1  # Linear equation

# Plot the curve
plt.plot(x, y, label="y = 2x + 1", color="blue")

# Add labels and title
plt.xlabel("x")
plt.ylabel("y")
plt.title("Linear Function: y = 2x + 1")

# Show the legend and plot
```

```
plt.legend()
plt.grid(True)
plt.show()
```




**Interpretation:**

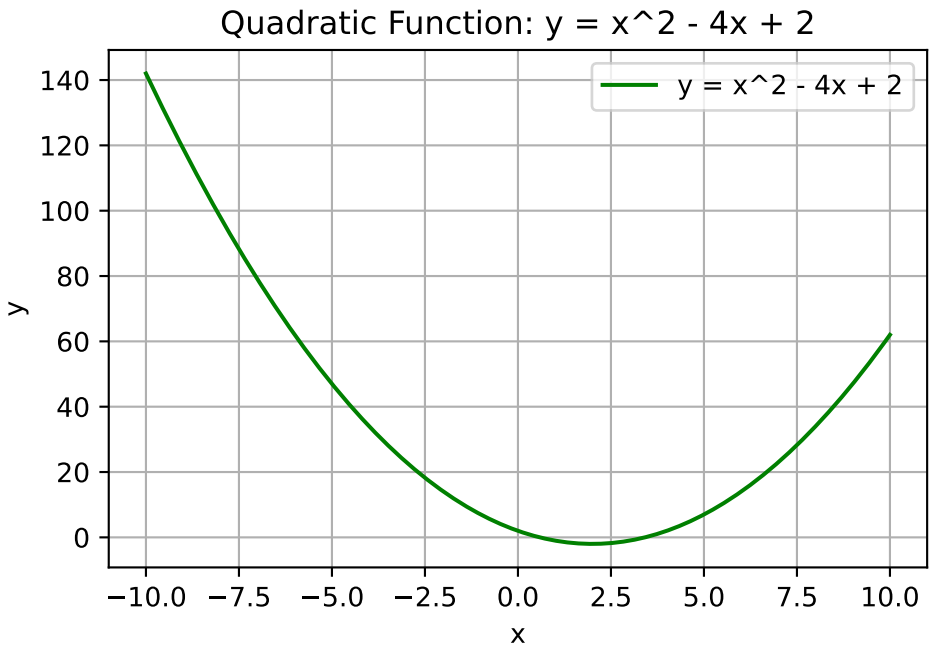
- `np.linspace()` generates a range of `x` values between -10 and 10.
- We apply the equation `y = 2x + 1` to calculate `y` for each `x`.
- The plot is displayed with a grid and labeled axes.

### 2. Plotting a Quadratic Function

We will plot a quadratic function of the form: $ax^2 + bx + c$

**Example: Plot** $y = x^2 - 4x + 2$

```python
import matplotlib.pyplot as plt
import numpy as np

# Data for the plot
x = np.linspace(-10, 10, 400)  # Generate 400 values from -10 to 10
y = x**2 - 4 * x + 2  # Quadratic equation

# Plot the curve
plt.plot(x, y, label="y = x^2 - 4x + 2", color="green")

# Add labels and title
plt.xlabel("x")
plt.ylabel("y")
plt.title("Quadratic Function: y = x^2 - 4x + 2")

# Show the legend and plot
plt.legend()
plt.grid(True)
plt.show()
```




**Interpretation:**

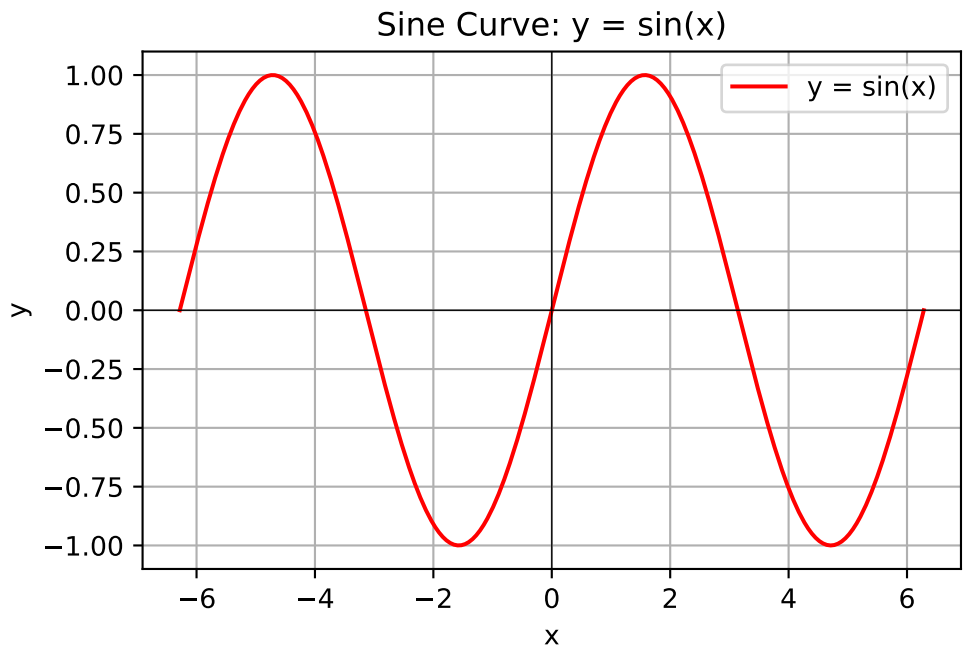
- This is a simple quadratic function plot, showing a parabolic curve.
- The `x**2` expression squares each `x` value to calculate `y`.

### 3. Plotting a Sine Curve

Sine curves are periodic and are defined as: $y = \sin(x)$

**Example: Plot** $y = \sin(x)$

```
import matplotlib.pyplot as plt
import numpy as np

# Data for the plot
x = np.linspace(-2 * np.pi, 2 * np.pi, 400)  # Generate values from -2  to 2
y = np.sin(x)  # Sine function

# Plot the curve
plt.plot(x, y, label="y = sin(x)", color="red")

# Add labels and title
plt.xlabel("x")
plt.ylabel("y")
plt.title("Sine Curve: y = sin(x)")

# Show the legend and plot
plt.legend()
plt.grid(True)
plt.axhline(0, color='black',linewidth=0.5)  # Add horizontal line at y=0
plt.axvline(0, color='black',linewidth=0.5)  # Add vertical line at x=0
plt.show()
```




**Interpretation:**

- The `np.sin()` function generates the sine of each `x` value, creating the periodic wave.
- We plot the function over a range of `x` values between `-2` and `2` .

### 4. Plotting a Parametric Curve

Parametric curves are plotted using a parameter (often denoted as `t`) that affects both `x` and `y`.

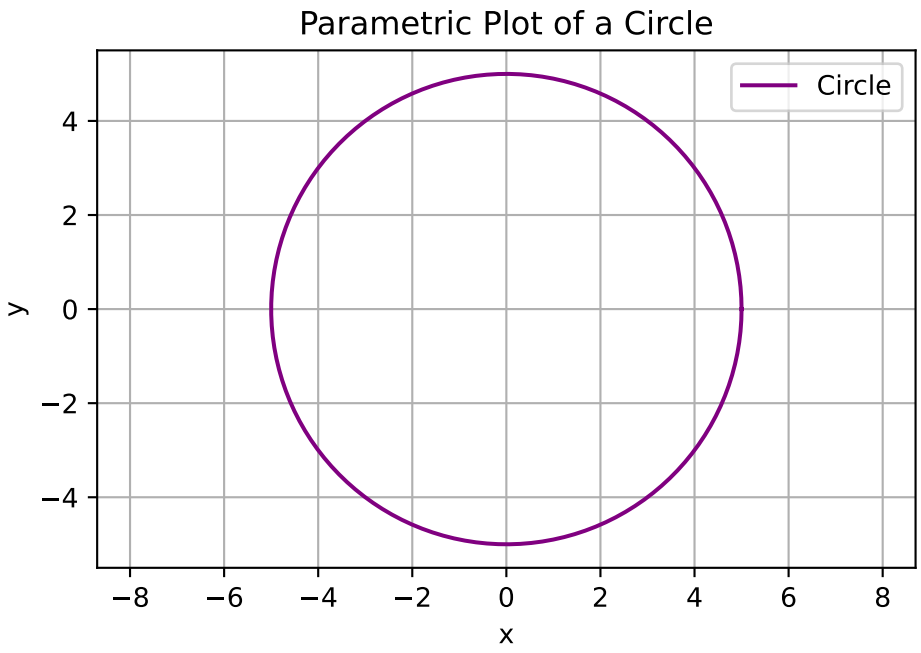
**Example: Plot the parametric equation of a circle**

$$x = r\cos t$$

$$y = r\sin t$$

for $t$ in the range $[0, 2\pi]$, where $r$ is the radius.

```python
import matplotlib.pyplot as plt
import numpy as np

# Data for the plot
t = np.linspace(0, 2 * np.pi, 400)  # Generate values of t from 0 to 2
r = 5  # Radius of the circle
x = r * np.cos(t)  # Parametric equation for x
y = r * np.sin(t)  # Parametric equation for y

# Plot the curve
plt.plot(x, y, label="Circle", color="purple")

# Add labels and title
plt.xlabel("x")
plt.ylabel("y")
plt.title("Parametric Plot of a Circle")

# Make axes equal to preserve the aspect ratio
plt.axis('equal')
plt.legend()
plt.grid(True)
plt.show()
```



**Interpretation:**

- The `np.cos()` and `np.sin()` functions generate the x and y coordinates of the circle, respectively.
- `plt.axis('equal')` ensures that the scale of the x and y axes is the same, preserving the circular shape.

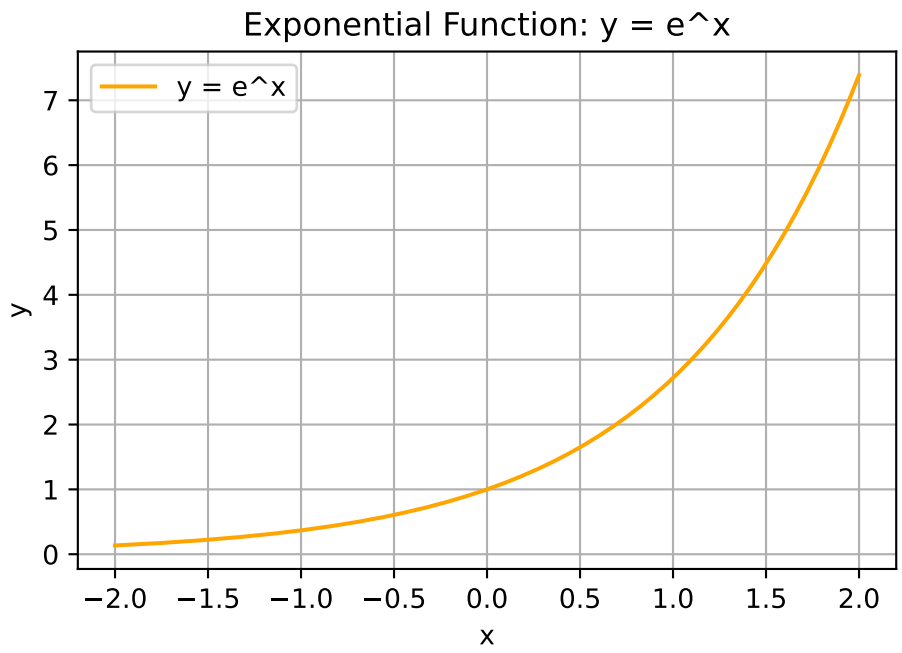
### 5. Plotting an Exponential Function

Exponential functions grow rapidly and are often used in growth models: $y = e^x$

where $e$ is Euler's number (approximately 2.718).

**Example: Plot $y = e^x$**

```
import matplotlib.pyplot as plt
import numpy as np

# Data for the plot
x = np.linspace(-2, 2, 400)  # Generate values from -2 to 2
y = np.exp(x)  # Exponential function

# Plot the curve
plt.plot(x, y, label="y = e^x", color="orange")

# Add labels and title
plt.xlabel("x")
plt.ylabel("y")
plt.title("Exponential Function: y = e^x")

# Show the legend and plot
plt.legend()
plt.grid(True)
plt.show()
```




**Interpretation:**

- The `np.exp()` function computes the exponential of each `x` value.
- This results in a rapidly increasing curve as `x` increases.

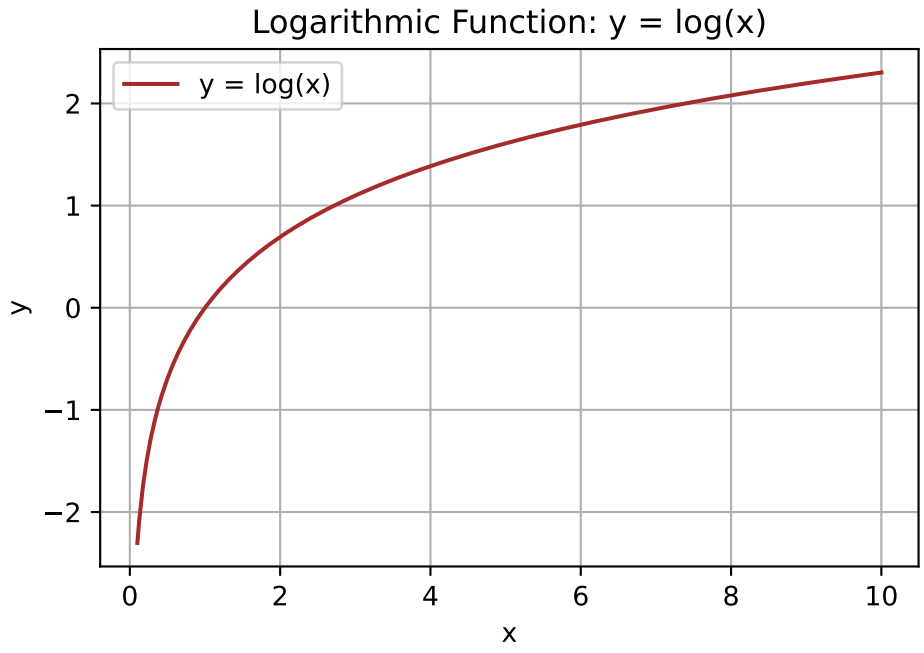
### 6. Plotting a Logarithmic Function

Logarithmic functions grow slowly as `x` increases: $y = \log(x)$ for $x > 0$.

**Example: Plot $y = \log(x)$**

```
import matplotlib.pyplot as plt
import numpy as np

# Data for the plot
x = np.linspace(0.1, 10, 400)  # Generate values from 0.1 to 10
y = np.log(x)  # Logarithmic function

# Plot the curve
plt.plot(x, y, label="y = log(x)", color="brown")
```

```
# Add labels and title
plt.xlabel("x")
plt.ylabel("y")
plt.title("Logarithmic Function: y = log(x)")

# Show the legend and plot
plt.legend()
plt.grid(True)
plt.show()
```




**Interpretation:**

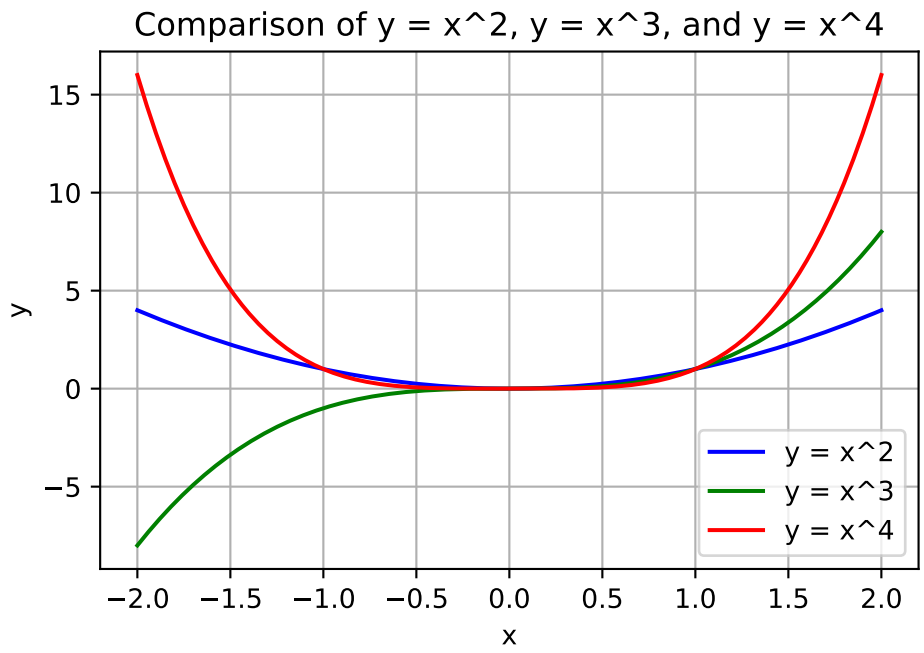
- The `np.log()` function computes the natural logarithm of each `x` value.
- The function grows slowly as `x` increases, and tends toward negative infinity as `x` approaches 0.

### 7. Multiple Curves on One Plot

You can plot multiple functions on the same axes to compare them.

**Example: Plot $y = x^3, y = x^3$, and $y = x^4$ on the same graph.**

```
import matplotlib.pyplot as plt
import numpy as np

# Data for the plot
x = np.linspace(-2, 2, 400)
y1 = x**2
y2 = x**3
y3 = x**4

# Plot multiple curves
plt.plot(x, y1, label="y = x^2", color="blue")
plt.plot(x, y2, label="y = x^3", color="green")
plt.plot(x, y3, label="y = x^4", color="red")

# Add labels and title
plt.xlabel("x")
plt.ylabel("y")
plt.title("Comparison of y = x^2, y = x^3, and y = x^4")

# Show the legend and plot
plt.legend()
plt.grid(True)
plt.show()
```




**Interpretation:**

- The plot shows how the functions $y = x^2, y = x^3$ , and $y = x^4$ differ in their growth patterns.

Curve plotting is a fundamental skill in mathematical and scientific computing. Python's `matplotlib` library makes it easy to create various types of plots, including linear, quadratic, trigonometric, and parametric curves. By learning to use `matplotlib` effectively, you can visualize mathematical functions and gain insights into their behavior.