Discrete Structures!

CMPSC 102

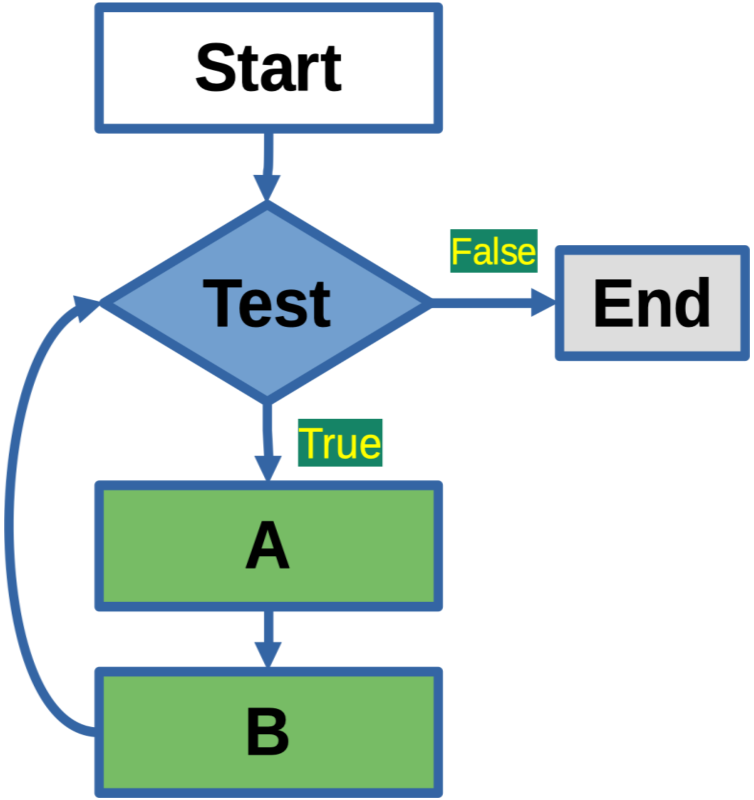
ALLEGHENY COLLEGE

Key Questions and Learning Objectives

- How do I use **iteration** and **conditional logic** in a Python program to perform computational tasks like processing a file's contents and mathematical tasks like using Newton's method to approximate the square root of a number?

- To remember and understand some discrete mathematics and Python programming concepts, setting the stage for exploring discrete structures.

Loops for Iteration - A loop is a way to reuse code blocks



The While Loop

```
index = 0
while (index < 10): # condition
        print(f"Count :{index}")
        index += 1 # add one to index
```

Output

```
Count :0
Count :1
Count :2
Count :3
Count :4
Count :5
Count :6
Count :7
Count :8
Count :9
```

The for … in range() Loop

```python
for index in range(10):
        print(f"Count :{index}")
```

Output

```
Count :0
Count :1
Count :2
Count :3
Count :4
Count :5
Count :6
Count :7
Count :8
Count :9
```

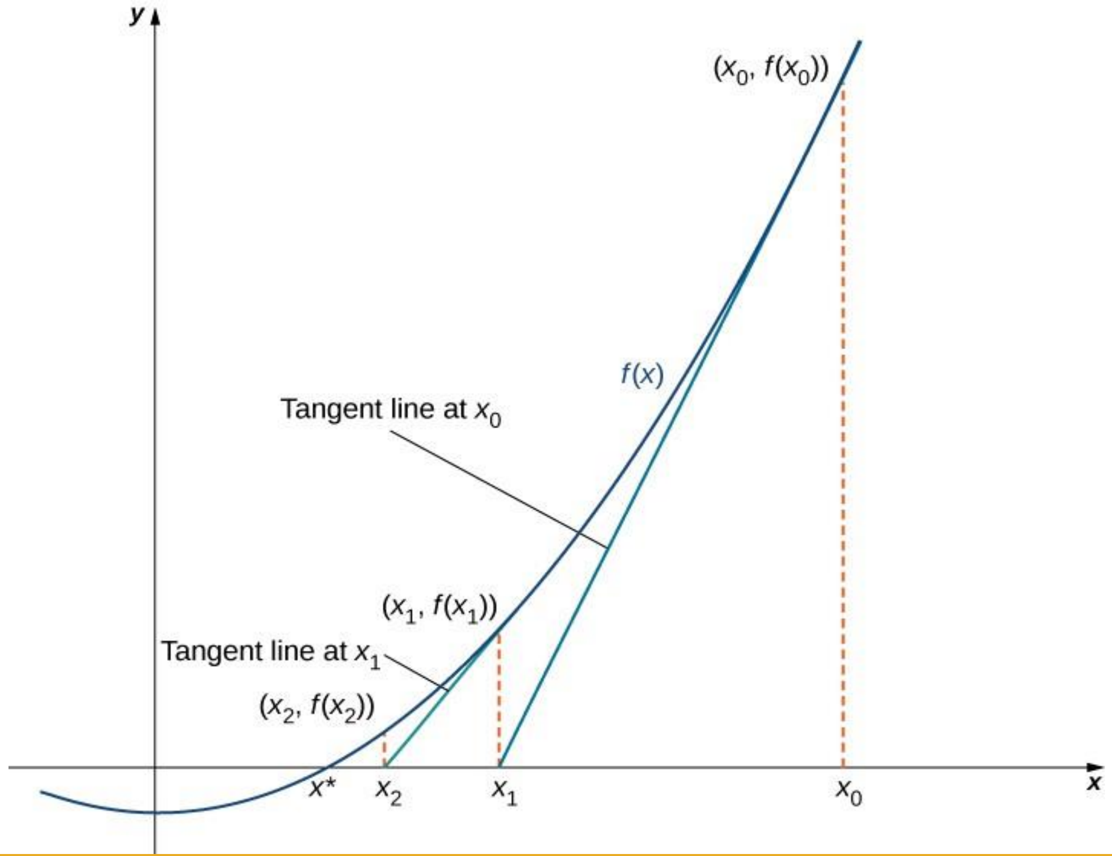
Square Roots - Mathematical loops to find quadruple roots

How to compute : $\sqrt{x}$?

Method

The function initializes the guess for the square root and iteratively refines it using an approximation formula until the approximation is within the specified tolerance.

Newton's Method - Mathematical loops to find square roots



The While Loop Application - Finding a Square Root

```
n = 4
guess = 1.0
while abs(n - guess*guess) > 0.0001:
        guess = guess - (guess*guess - n)/(2*guess)
        print(f"n = {n} : guess = {guess}")
```

- Iteratively **guesses** the square root until **within tolerance**
- The while loop uses ‘abs‘ for computing an absolute value
- This loop computes the root as 2.000000092922294 7
- The math.sqrt(n) function confirms this approximation!
- Any questions about this way to approximate a square root?

The While Loop Application - Finding a Square Root

```python
n = 4
guess = 1.0
while abs(n - guess*guess) > 0.0001:
        guess = guess - (guess*guess - n)/(2*guess)
        print(f"n = {n} : guess = {guess}")
```

Output

```
n = 4 : guess = 2.5
n = 4 : guess = 2.05
n = 4 : guess = 2.000609756097561
n = 4 : guess = 2.0000000929222947
```

The For Loop Application - Finding a Square Root

```
n = 4
guess = 1.0
for i in range(5):
        abs(n - guess*guess) > 0.0001
        guess = guess - (guess*guess - n)/(2*guess)
        print(f"n = {n} : guess = {guess}")
```

Output

```
n = 4 : guess = 2.5
n = 4 : guess = 2.05
n = 4 : guess = 2.000609756097561
n = 4 : guess = 2.0000000929222947
```

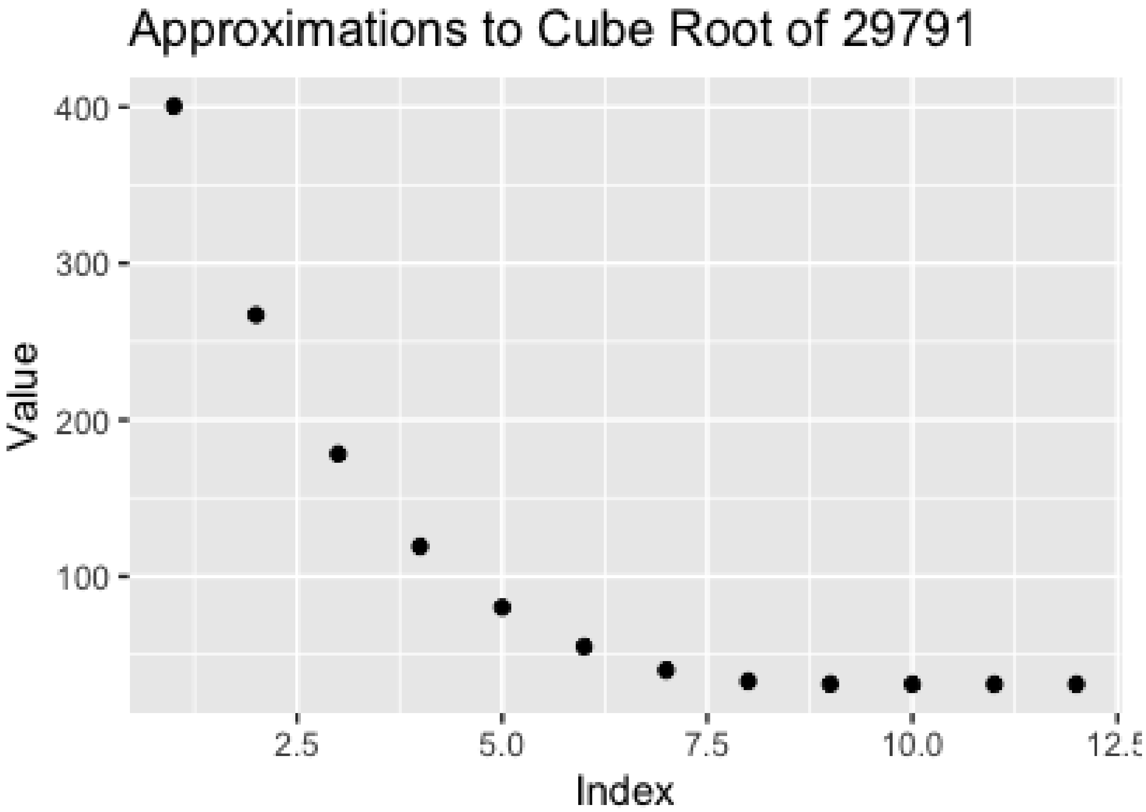
Cube Roots - Mathematical loops to find cube roots

How to compute : $\sqrt[3]{x}$?

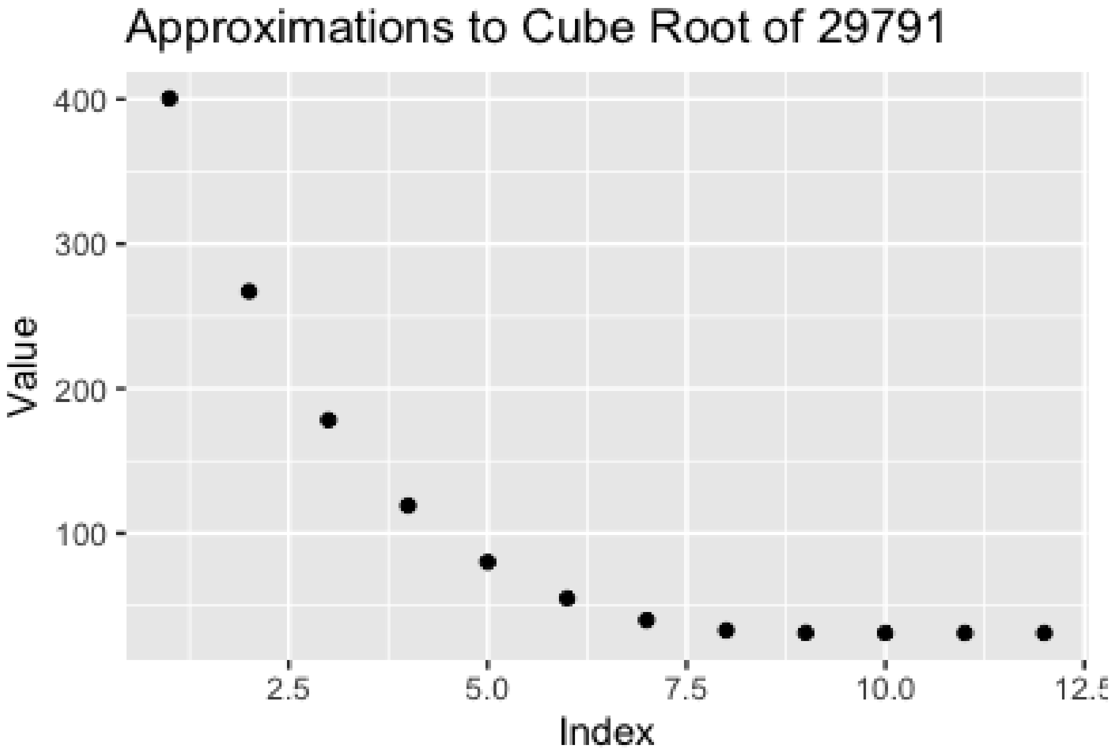
Method

The function initializes the guess for the cube root and iteratively refines it using an approximation formula until the approximation is within the specified tolerance.

Cube Roots - Approximations to find cube roots; result = 31



Cube Roots - Approximations to find cube roots; result = 31



400.546666666667,
267.0930063983859,
178.2012041240575,
119.11351331724008,
80.10891816753315,
54.95334366560159,
39.923893861643855,
32.846072628479256,
31.101812384645754,
31.00033292113973,
31.00000000357532,
31.0

Cube Roots - Code

```
def cube_root_approximation(number, tolerance=1e-6):
        # Initial guess for the cube root
        # guess = number / 2.0 # one way to start
        guess = 5
        # Iterate until the approximation
        # is within the specified tolerance
        while abs(guess**3 - number) > tolerance:
                # Update the guess using the approximation formula
                guess = (2 * guess + number / (guess**2)) / 3.0
                print(f" guess = {guess}")
        return guess

# Example: Calculate the cube root of 29791 i
input_number = 29791
result = cube_root_approximation(input_number)

# Display the result
print(f"The cube root of {input_number}")
print(f" is approximately: {result}")
```

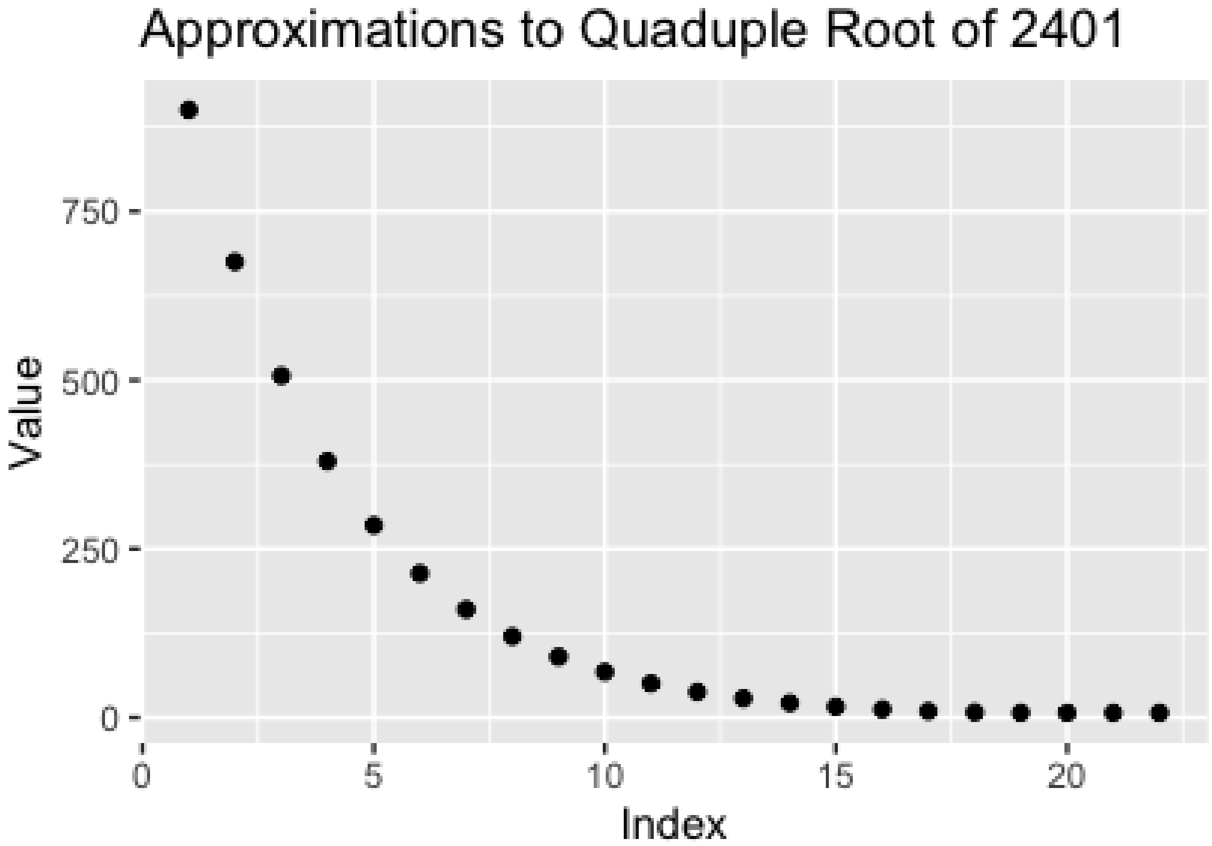
Cube Roots - Mathematical loops to find cube roots

How to compute : $\sqrt[4]{x}$?

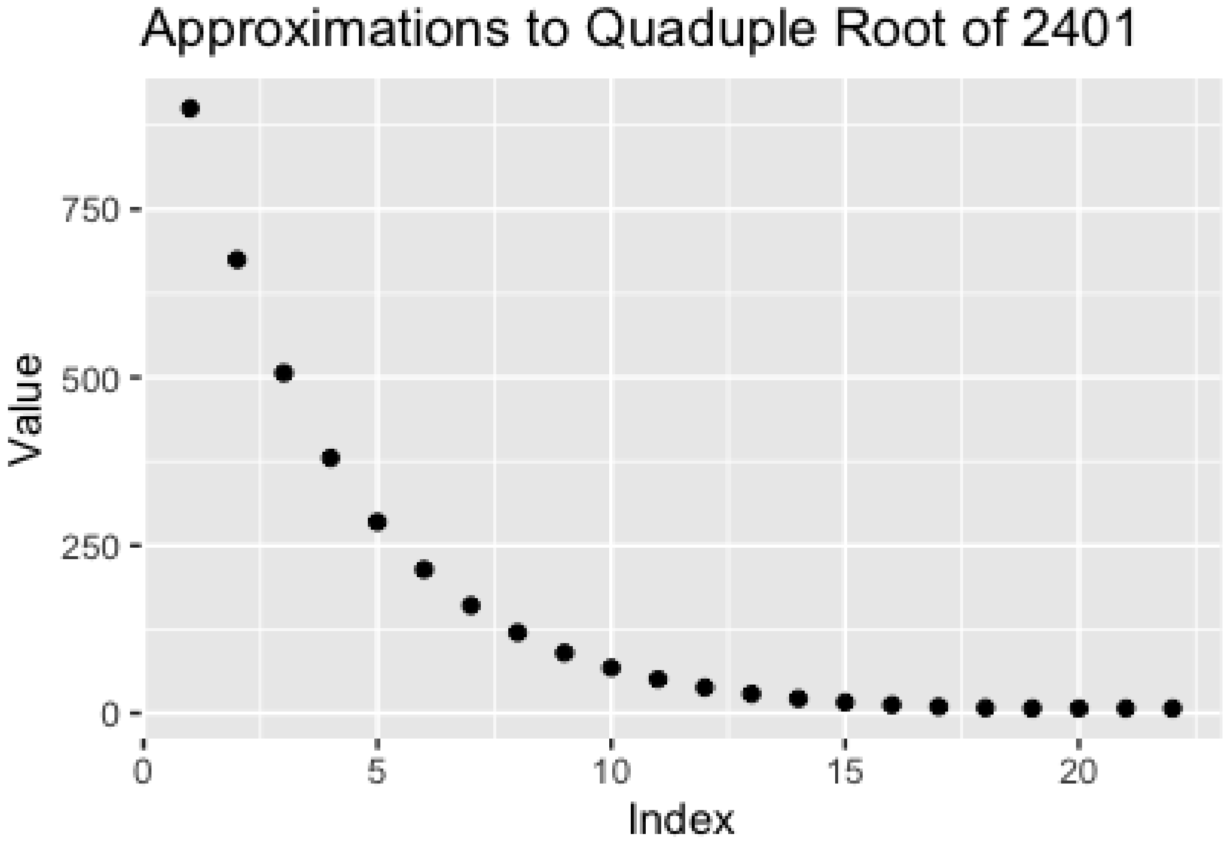
Method

The function initializes the guess for the quadruple root and iteratively refines it using an approximation formula until the approximation is within the specified tolerance.

Quadruple Roots - Approximations to find quadruple roots; result = 7



Quadruple Roots - Approximations to find quadruple roots; result = 7



900.375000346933,
675.2812510825596,
506.4609402612170 5,
379.8457098164694,
284.8842933147822,
213.66324594739956,
160.24749599845413,
120.1857678662979 1,
90.13967165836334,
67.60557331037666,
50.70612258852655,
38.03419610727984,
28.5365566959587,
21.428247703754217,
16.132191727838684,
12.242116142093641,
9.508748977118197,
7.82973307939813,
7.122821698405513,
7.00314043464742,
7.000002111777238,
7.000000000000956

Quadruple Roots - Code

```
def quadruple_root_approximation(number, tolerance=1e-6):
        # Initial guess for the fourth root
        guess = number / 2.0
        # Iterate until the approximation
        # is within the specified tolerance
        while abs(guess**4 - number) > tolerance:
                # Update the guess using the approximation formula
                guess = (3 * guess + number / (guess**3)) / 4.0
                print(f" guess = {guess}")
        return guess

# Example: Calculate the fourth root of 2401
input_number = 2401
result = quadruple_root_approximation(input_number)

# Display the result
print(f"The fourth root of {input_number}")
print(f" is approximately: {result}")
```

Quadruple Roots – The Problem Defined

To Solve: $x^2 + 3x - 4 = 0 \quad (1)$

Want to have roots

$$x_1 = ?\ and\ x_2 = ?$$

Quadruple Roots – The Problem Defined

Quadratic Equation:

$$ax^2 + bx + c = 0 \quad (2)$$

Quadratic Formula

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \quad (3)$$

Special Note

Note the $x_{1,2}$ to imply that there are two solutions (i.e., x_1 and x_2) to find for a second degree equation as observed from the x^2.

Quadruple Roots – The Problem Defined

```
def calc_quad_eqn_roots(
a: float, b: float, c: float) -> float:
        """Calculate roots of quadratic equation."""
        D = (b * b - 4 * a * c) ** 0.5
        x_one = (-b + D) / (2 * a)
        x_two = (-b - D) / (2 * a)
        return x_one, x_two

print(f"{calc_quad_eqn_roots(1,2,1)}")
```

- Three floating-point inputs: a, b, and c
- Two floating-point outputs: x_{one} and x_{two}
- How does it calculate the roots of a quadratic equation?