Discrete Structures: Programming Constructs CMPSC 102

Oliver BONHAM-CARTER

Fall 2022
Week 3
Slides 02

Let's Discuss

Key Questions

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?

Learning Objectives

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

Loops for Iteration

A loop is a way to reuse code blocks

Discrete Structures: Programming Constructs CMPSC 102

Oliver BONHAM-CARTER

Let's Discuss

Loops

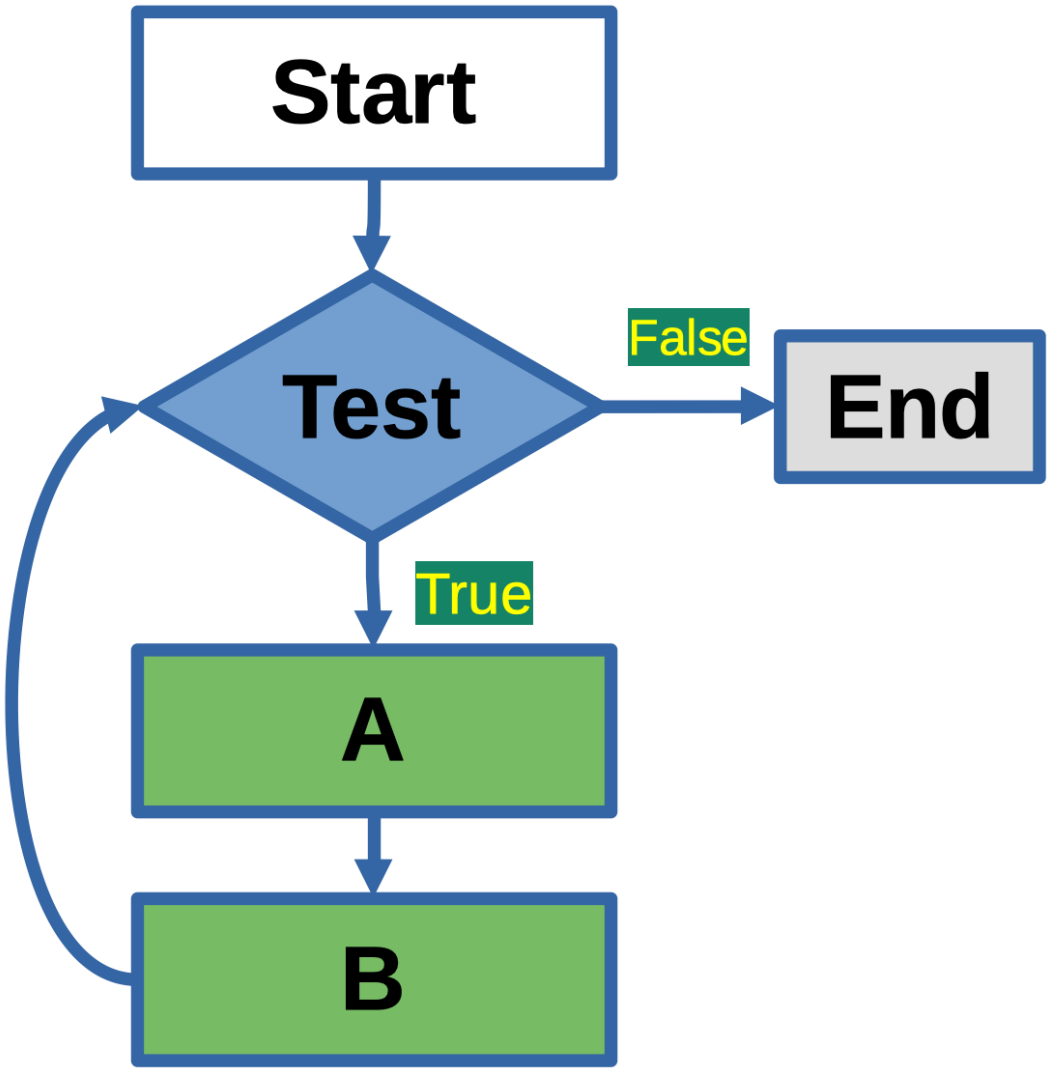
While

For

Newton's Method

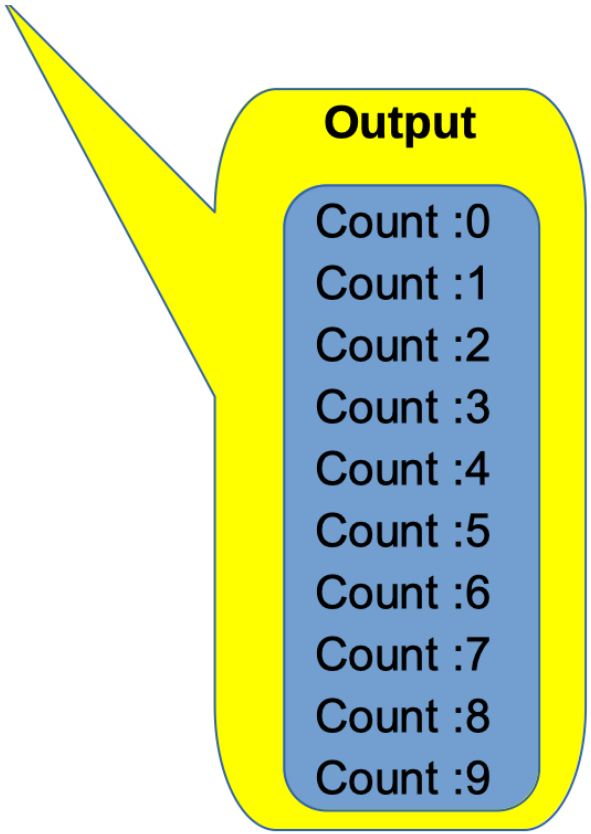
Quadratic Roots

Solutions



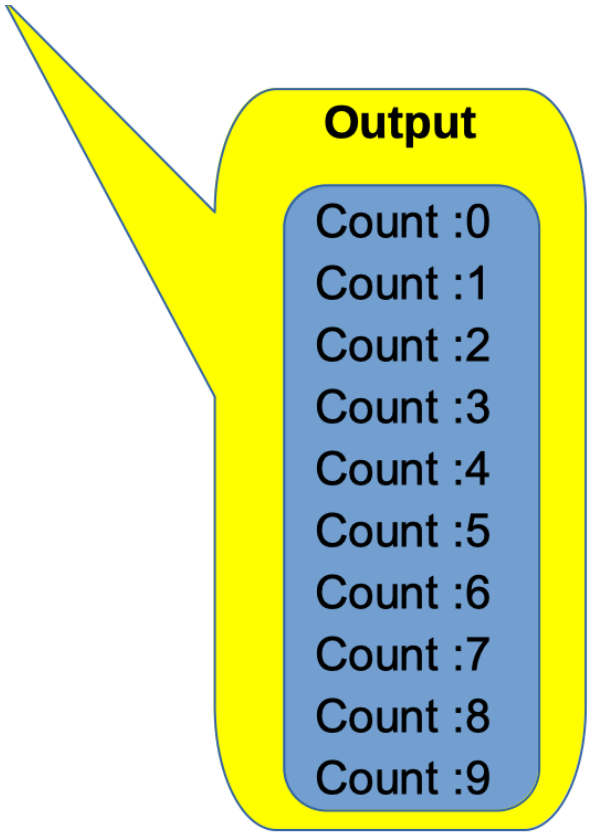
The While Loop

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



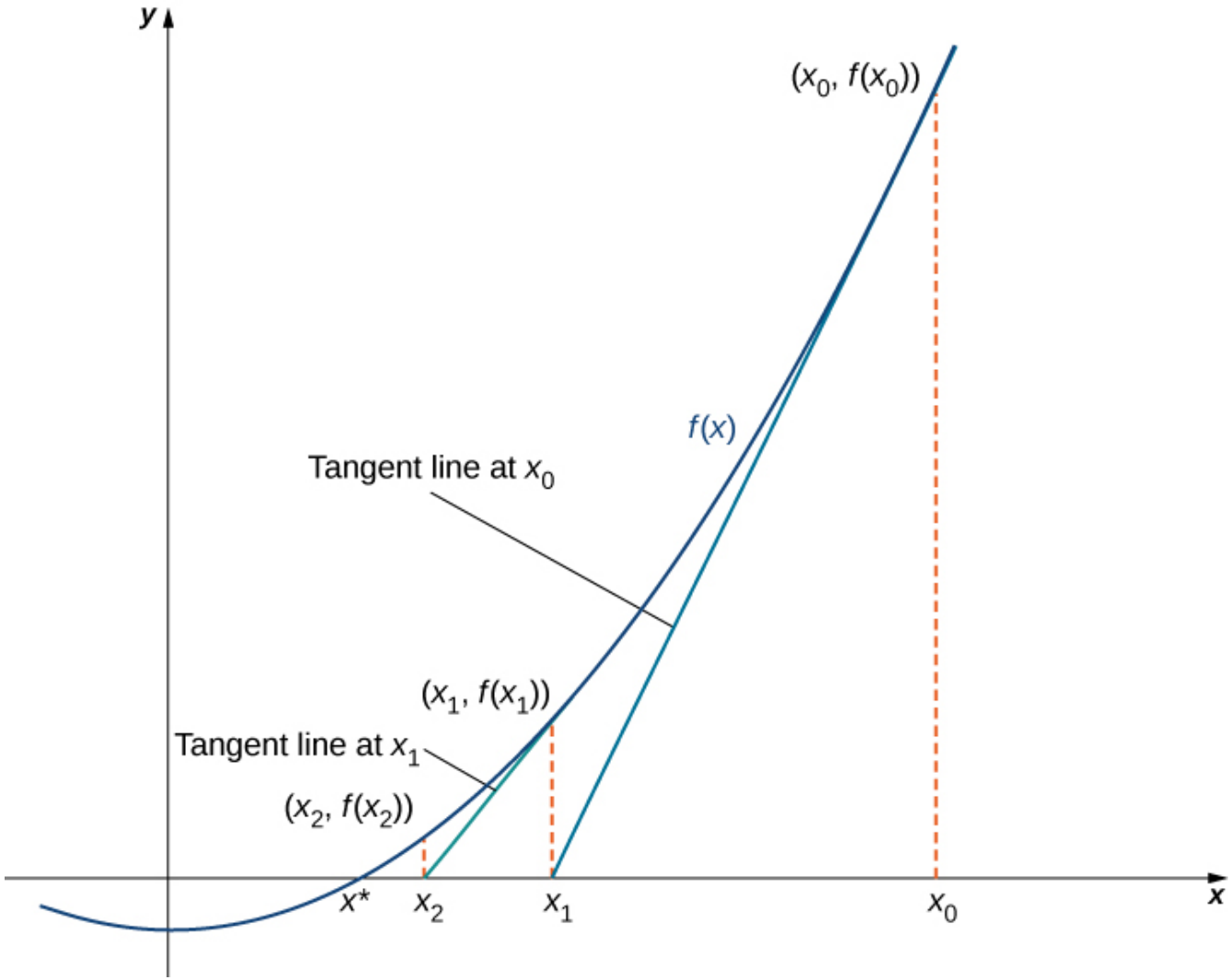
The for ... in range() Loop

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



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 2.000000929222947
- 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

```
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 = 2.000000929222947
```

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 = 2.000000929222947
```

Quadratic Root Calculation

(Back to where we were ...)

Quadratic Equation

$$ax^2 + bx + c = 0 \tag{1}$$

Quadratic Formula

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{2}$$

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.

Our Solution From Class

Function: `main()`

```
def main():
    """Docstring: driver of the program;
       deals with user"""
    print("Quadratic equation solver:")
    print("Enter your a,b,c integer values below")

    # Get inputs from user
    prompt = " Enter a :"
    a_int = int(input(prompt))
    prompt = " Enter b :"
    b_int = int(input(prompt))
    prompt = " Enter c :"
    c_int = int(input(prompt))
```

Our Solution From Class

Continuing Function `main()`

```
    # want to check that values are integers
    print(f" the types for a,b,c resp are:
    {type(a_int)}, {type(b_int)}, {type(c_int)}")
    print("preparing root calculation ...")
    # call quadratic formula solver
    root1,root2 = calcQuadratic(a_int, b_int, c_int)
# end of main()
```

Our Solution From Class

Function quadSolver()

```
def quadSolver(a,b,c):
"""function to apply quadratic
   forumula to find roots"""
a=int(a)
b=int(b)
c=int(c)
print(f"\tReceived a={a}, b={b}, c={c}")

# sqrt of discriminant
D = (b * b - 4 * a * c) ** 0.5
x_one = (-b + D) / (2 * a) # first root
x_two = (-b - D) / (2 * a) # second root
return x_one, x_two
# end of quadSolver()
```

Another Solution

Function `main()`

```
def main(
    """Driver Function"""
    a: float = typer.Option(1),
    b: float = typer.Option(2),
    c: float = typer.Option(2)
    console.print(f"Calculating roots with:")
    console.print(f"   a = {a}")
    console.print(f"   b = {b}")
    console.print(f"   c = {c}")
    x_one, x_two =
           rootfind.
    calc_quad_eqn_roots(a, b, c)
    console.print(f"   x_one = {x_one}")
    console.print(f"   x_two = {x_two}")
```

Another Solution

Function `calc_quad_eqn_roots()`

```
def calc_quad_eqn_roots(a: float, b: float, c: 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
```

- 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?

- **How do we test functions to ensure correctness?**

Creating Solutions

File: squareArea.py

Function `squareArea()`

Discrete Structures: Programming Constructs CMPSC 102

Oliver BONHAM-CARTER

Let's Discuss

Loops

While

For

Newton's Method

Quadratic Roots

Solutions

```
def squareArea(s: float ) -> float:
    return s*s # area of square is s*s
# end of squareArea()
```

What inputs s are acceptable?

- integers?
- floats?
- booleans?
- strings?
- imaginary numbers? $(1 + 3j)$

File: squareArea.py

Function `main()`

```
def main(in_str):
    # passes
    sideLength = 5
    print(f"Length {sideLength},
                area: {squareArea(sideLength)}")

    # does not pass since some values are not ints or floats
    testValues_list =[2,0,-3,2 + 5j, True, "radius"]
    for val in testValues_list:
        print(f"Length {val}, area: {squareArea(val)}")
# end main()
```

Can we calculate with the values in `testValues_list`?

File: squareArea.py

Initiation code

```
import sys
if __name__ == '__main__':
main(sys.argv[1:])

main()
```