CS1010S

Tutorial 2: Recursion and Iteration

Nicholas Russell Saerang (russellsaerang@u.nus.edu)

Admin

Next Monday is **CNY Holiday**.

We will have make-up tutorial via Zoom. Please poll accordingly on the Telegram chat!

Flowchart Advice

- Flowchart is meant to be a **CLEAR & UNAMBIGUOUS** instructions.
- Please follow the convention taught in lecture 1!!
- And it will be tested, so practice it

Table of contents

Key Concepts

Variable Scope, Control Flow

Variable Scope

Global, local, namespaces

Variable Scope
Global and Local Scopes

```
def darth_vader(son):
    # local
    print(”I am your father, ” + son)
# global
```

Variable Scope
Local Scope

```
def darth_vader(son):
    print(”I am your father, ” + son)

>>> darth_vader("Luke") # Luke
I am your father, Luke
```

Variable Scope
Local Scope

```
son = "Abraham"

def darth_vader(son):
    print(”I am your father, ” + son)

>>> darth_vader("Luke")
```

Variable Scope
Local Scope

```
son = "Abraham"

def darth_vader(son):
    print(”I am your father, ” + son)

>>> darth_vader("Luke")   # Luke
I am your father, Luke
```

Variable Scope
Local Scope

```
son = "Abraham"

def darth_vader(son):
    son = "Lucas"
    print(”I am your father, ” + son)

>>> darth_vader("Luke")
```

Variable Scope
Local Scope

```
son = "Abraham"

def darth_vader(son):
    son = "Lucas" # assignment
    print(”I am your father, ” + son)

>>> darth_vader("Luke")   # Lucas
I am your father, Lucas
```

Variable Scope
Local Scope

```
son = "Abraham"

def darth_vader(son):
    print(”I am your father, ” + son)
    son = "Lucas"

>>> darth_vader("Luke")
```

Variable Scope
Local Scope

```
son = "Abraham"

def darth_vader(son): # assignment
    print(”I am your father, ” + son)
    son = "Lucas"

>>> darth_vader("Luke")   # Luke
I am your father, Luke
```

Variable Scope
Local Scope

```
son = "Abraham"

def darth_vader():
    print(”I am your father, ” + son)
    son = "Lucas"

>>> darth_vader()
```

Variable Scope
Local Scope

```
son = "Abraham"

def darth_vader():
    print(”I am your father, ” + son)
    son = "Lucas"

>>> darth_vader()
UnboundLocalError
```

Variable Scope
Global Scope

```
son = "Abraham"

def darth_vader():
    print("I am your father, " + son)

>>> darth_vader()
```

Variable Scope
Global Scope

```
son = "Abraham" # Assignment

def darth_vader():
    print(”I am your father, ” + son)

>>> darth_vader()  # Abraham
I am your father, Abraham
```

Variable Scope
Global Scope

```
def darth_vader():
    print(”I am your father, ” + son)

son = "Abraham" # Assignment

>>> darth_vader()  # Abraham
I am your father, Abraham
```

Variable Scope
Test Yourself

```
# Tip: search from innermost to outermost (global)
def f1():
    a = 40
    def f2():
        b = 40
        def f3():
            a = 50
            print(a, b, c)
        print(a, b, c)
        f3()
    print(a, c) # b is not defined here!
    f2()

c = 60
f1()
```

Control Flow

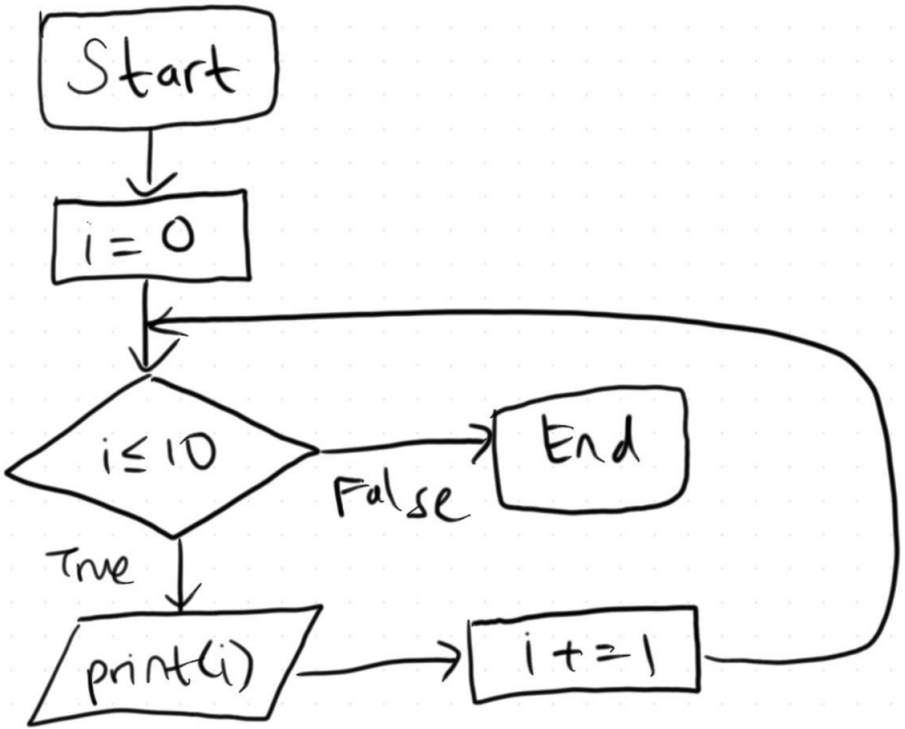
Iteration / Recursion, While

Control Flow
Iteration/Recursion

- Used to repeat chunks of code

```
i = 0
print(i)
i += 1
print(i)
i += 1
print(i)
i += 1
print(i)
...
```

```
i = 0
while i <= 10:
    print(i)
    i += 1
```


Start
i = 0
i ≤ 10
False
End
True
print(i)
i += 1

Control Flow
Recursion

Recursive functions are functions that call other recursive functions without having yet returned a value.
(Always have a base case!!!)

```
def powertwo(i):
    if i == 0:
        return 1
    else:
        return powertwo(i-1) + powertwo(i-1)

# powertwo(1)
    # powertwo(0) => return 1
    # powertwo(0) => return 1
# return 1 + 1 = 2
```

Control Flow
Recursion

Recursive functions are functions that call other recursive functions without having yet returned a value.
(Always have a base case!!!)

```
def powertwo(i):
    if i == 0:
        return 1
    else:
        res = powertwo(i-1)
        return res + res

# powertwo(1)
    # powertwo(0) => return 1
# return 1 + 1 = 2
```

Control Flow
Iteration

Recursion can be elegant.

But iteration usually more efficient.

```
def powertwo(n):
    res = 1
    count = 0                    # powertwo(1)
    while count < n:             # ← return 2
        res = res + res
        count += 1
    return res
```

Control Flow
While loops

- While loops are more customizable
- Usually when number of iterations are unknown

```
def collatz_conjecture(i):
    while i != 1:
        if i%2 == 0:
            i = i//2
        else:
            i = 3*i+1
        print(i)

# 12 → 6 → 3 → 10 → 5 → 16 → 8 → 4 → 2 → 1 (9 iterations)
# 27 → 82 → 41 → ⋯ → 20 → 10 → ⋯ → 2 → 1 (111 iterations!)
```

Tutorial 2

recursion, iteration

Question 1: Sum of Even Factorials

Write a function **sum_even_factorials** that finds the sum of the factorials of the even numbers that are less than or equal to **n**, where **n ≥ 0**.

```
>>> sum_even_factorials (1)
1
>>> sum_even_factorials (3)
3
>>> sum_even_factorials (6)
747
```

Question 1: Sum of Even Factorials

```
def factorial(n): # Functional Abstraction!!!!
    result = 1
    i = 1
    while i < n+1:
        result *= i
        i += 1
    return result

def sum_even_factorials(n):
    if n == 0 :
        return factorial(1)
    if n % 2 != 0:
        return sum_even_factorials(n-1)
    else:
        return factorial(n) + sum_even_factorials(n-2)
```

Question 1: Sum of Even Factorials

```
def factorial(n):
    result = 1
    i = 1
    while i < n+1:
        result *= i
        i += 1
    return result
```

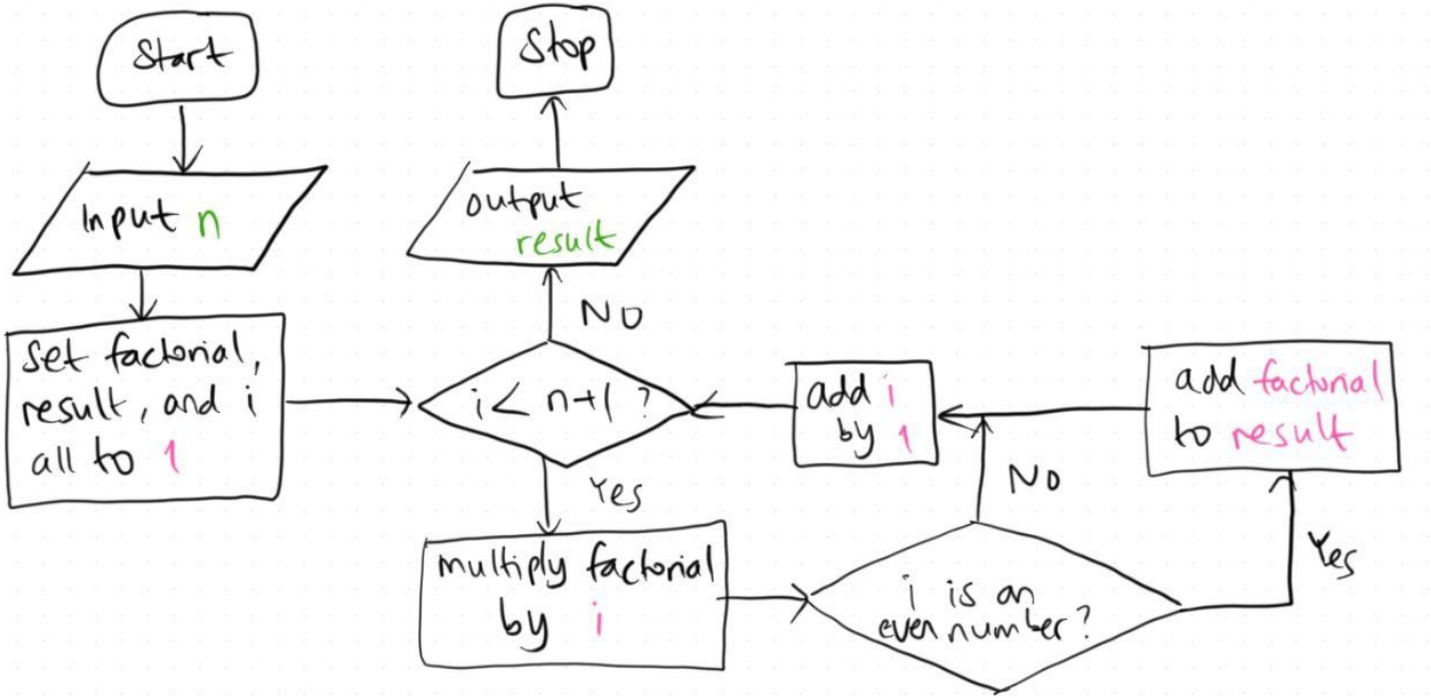
```
def sum_even_factorials(n):
    result = 0
    i = 0
    while i < n+1:
        if i % 2 != 0:
            i += 1
        else:
            result += factorial(i)
            i += 1
    return result
```

Functional Abstraction!

Question 1: Sum of Even Factorials

```
def sum_even_factorials(n): # combine both functions to one
    result = 1 # assuming n >= 0
    factorial = 1
    i = 1
    while i < n+1:
        factorial *= i # calculate the factorial of i
        if i % 2 == 0: # if i is even, add the factorial
            result += factorial
        i += 1
    return result
```

Question 1: Sum of Even Factorials



Question 3: Digit-counting

Implement a function that will return the number of digits in an integer. You can safely assume that the integers are non-negative and will not begin with the number 0, other than the integer 0 itself.

```
def number_of_digits(i):
    if i < 10:
        return 1
    else:
        return 1 + number_of_digits(i // 10)
```

Question 3: Digit-counting

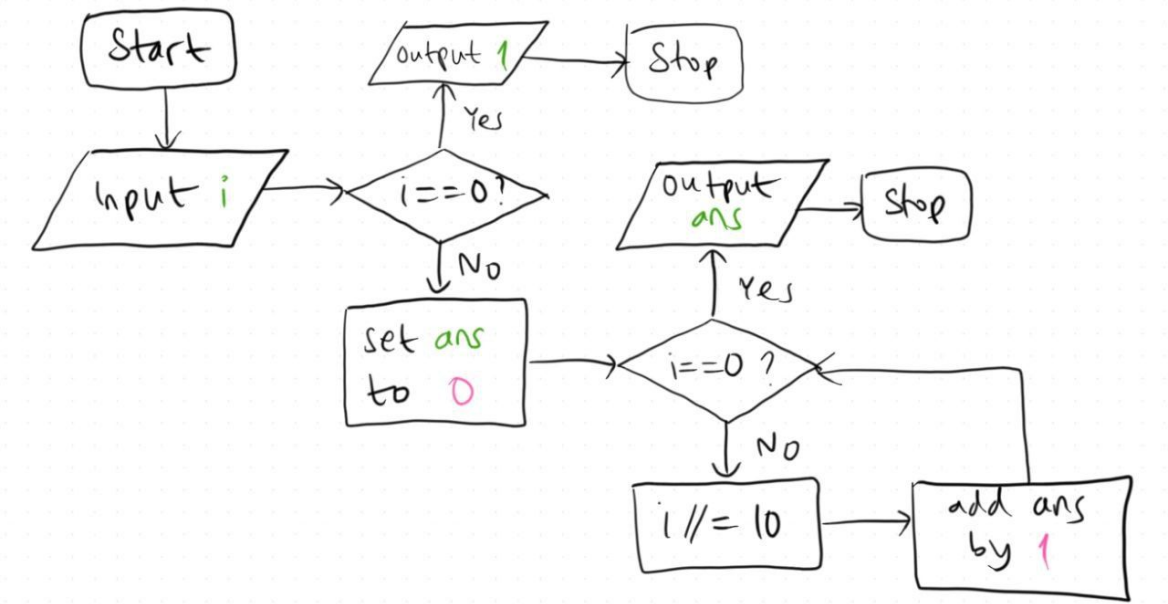
Implement a function that will return the number of digits in an integer. You can safely assume that the integers are non-negative and will not begin with the number 0, other than the integer 0 itself.

```
def number_of_digits(i):
    if i == 0:
        return 1
    ans = 0
    while (i > 0):
        i //= 10
        ans += 1
    return ans
```

```
def number_of_digits(i):
    return len(str(i))
```

Question 3: Digit-counting

Implement a function that will return the number of digits in an integer. You can safely assume that the integers are non-negative and will not begin with the number 0, other than the integer 0 itself.



Question 2: Recursion relation

$$f(n) = \begin{cases} n & n < 3 \\ f(n-1) + 2f(n-2) + 3f(n-3) & n \geq 3 \end{cases}$$

(a) Implement a function that computes f(n) by means of a recursive process.

We will discuss in the next tutorial.

Take your time to try this question!!

Especially, try figure out the iterative version yourself

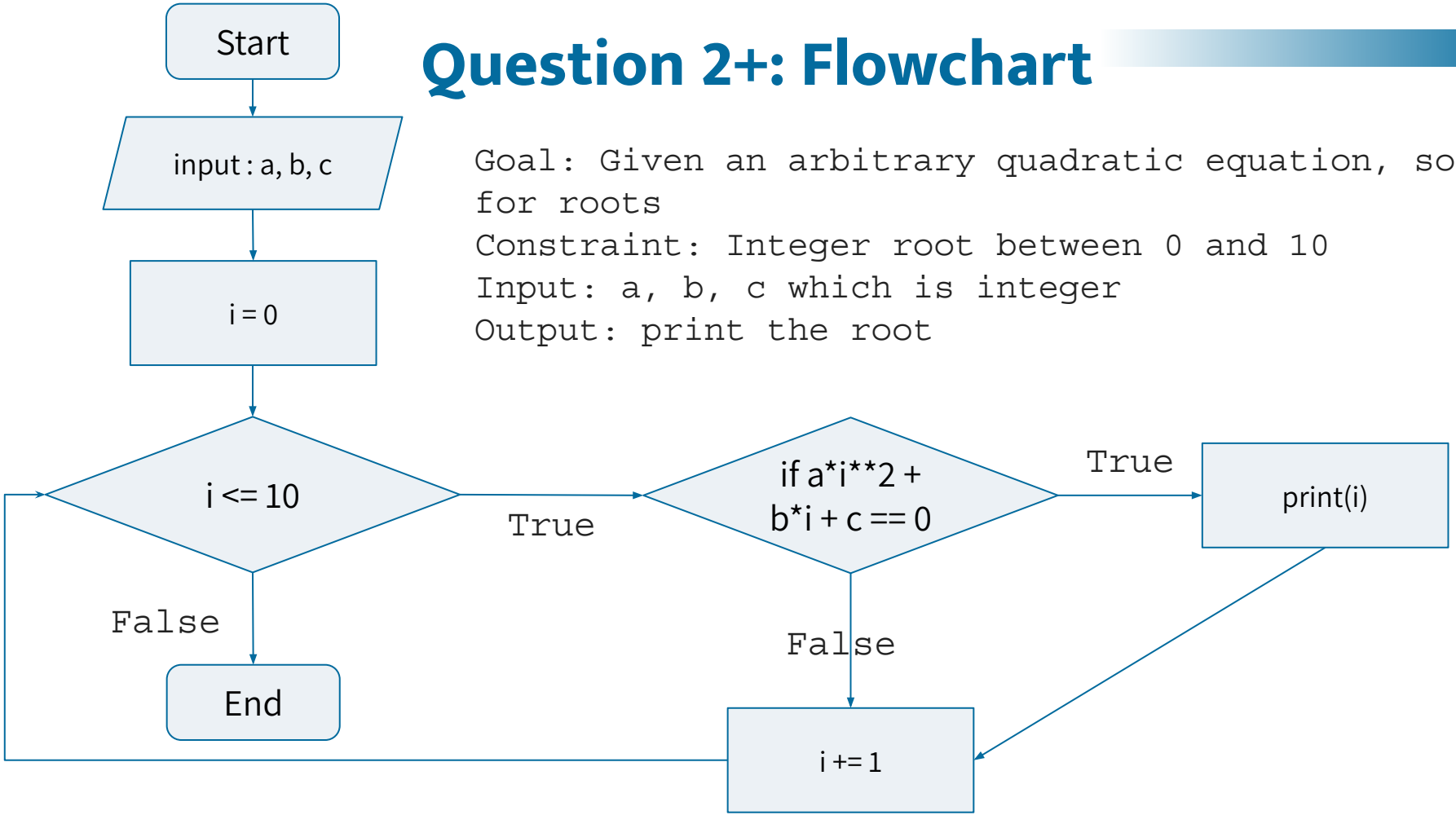
Question 2+: Flowchart

Let's design a root-finding algorithm for quadratic equation, **ax^2 + bx + c**, without using the closed-form solution, we only interested to integer solution between 0 to 10.

Goal: Given an arbitrary quadratic equation, solve for roots
Constraint: Integer root between 0 and 10
Input: a, b, c which is integer
Output: print the root

Question 2+: Flowchart

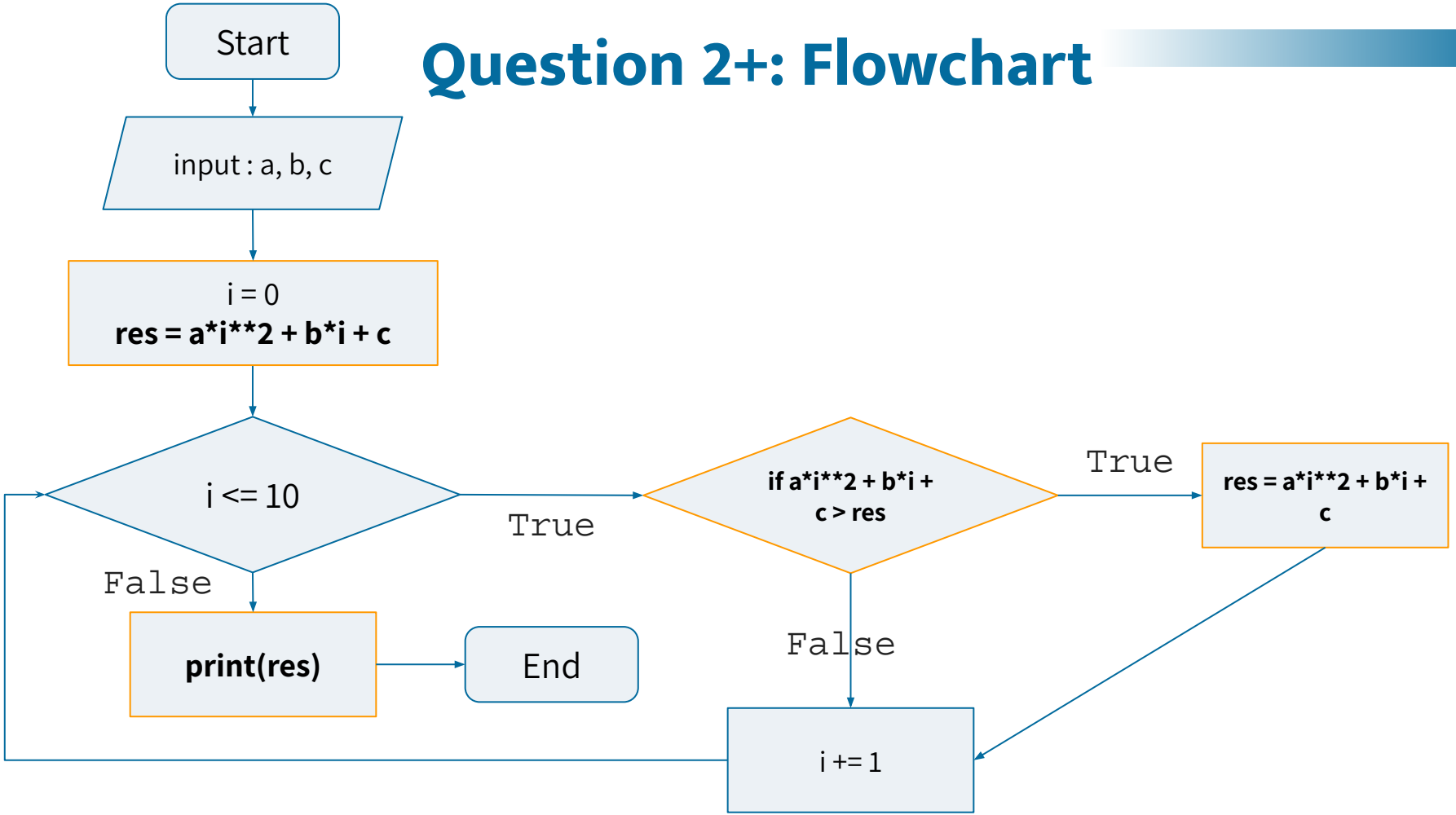
Goal: Given an arbitrary quadratic equation, solve for roots
Constraint: Integer root between 0 and 10
Input: a, b, c which is integer
Output: print the root



Question 2+: Flowchart

What if you are interested to find maximum value between 0 and 10. What should you change?

Question 2+: Flowchart



Extra Questions

Extra Questions

```
def weird_sum(n):
    if n == 0:
        return 0
    else:
        return n + weird_sum(n-2)

print(weird_sum(5))
```

Extra Questions

```
def weird_sum(n):
    if n == 0:
        return 0
    else:
        return n + weird_sum(n-2)

print(weird_sum(5))

# RecursionError: maximum recursion depth exceeded in
comparison
```

Extra Questions

```
def infinite_sum():
    res = 0
    n = 1
    factorial = 1
    while 1/n != 0:
        res += 1 / factorial
        n += 1
        factorial *= n
        print(res)
    return res

print(infinite_sum())
```

```
# If you have a SUPER GOOD COMPUTER
#
# It will stop when n reaches order
# of magnitude of (10 ** 1000)
#
# Because 1 / (10 ** 1000) = 0.0
#
# But usually your python will crash
# before reaching that
#
# However notice that this is NOT an
# infinite loop
```

The End