BIOS6642

# Iteration

**Fuyong Xing**

**Department of Biostatistics and Informatics**

**Colorado School of Public Health**

**University of Colorado Anschutz Medical Campus**

# Outline

- **The *while* statement**
- The *for* statement
- The *break* and *continue* statement
- Iteration examples

# The *while* statement

- Iteration or looping repeats the execution a sequence of code.
- Suppose we want to count to 5 by printing a number on each output line.

Codes

```
print(1)
print(2)
print(3)
print(4)
print(5)
```

Output

# The *while* statement

- Iteration or looping repeats the execution a sequence of code.
- Suppose we want to count to 5 by printing a number on each output line.

Codes

```
print(1)
print(2)
print(3)
print(4)
print(5)
```

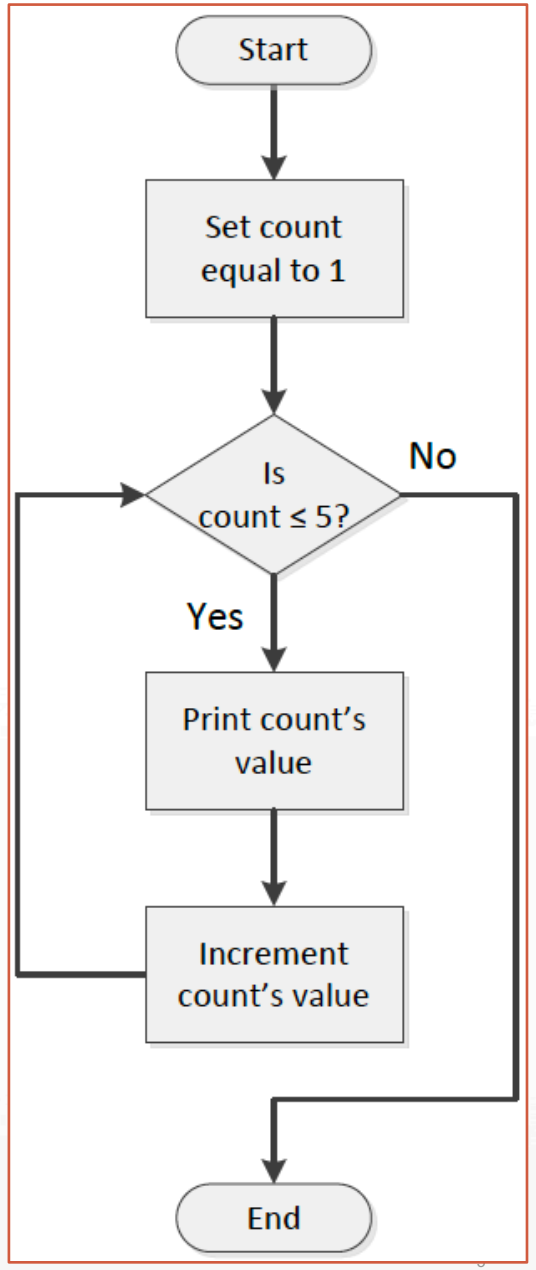
Output


- What if we want to count to 10000?

# The *while* statement

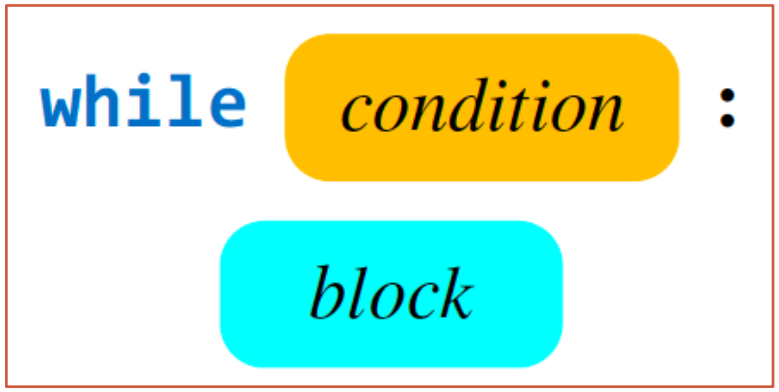
- The *while* statement

```
count = 1                   # Initialize counter
while count <= 5:           # Should we continue?
    print(count)            # Display counter, then
    count += 1              # Increment counter
```



# The *while* statement

- The *while* statement




- The reserved word *while* begins the *while* statement.
- The *condition* determines whether the body will be (or will continue to be) executed. A colon (:) must follow the condition.
- The *block* is a block of one or more statements to be executed as long as the condition is true. As a block, all the statements that comprise the block must be indented one level deeper than the line that begins the while statement.

# The *while* statement

- Definite loop: we know the exact number of iterations the loop will perform.

```
n = 1
while n <= 10:
    print(n)
    n += 1
```

# The *while* statement

- Indefinite loop: we cannot predict at any point during the loop's execution how many iterations the loop will perform.

```
done = False                    # Enter the loop at least once
while not done:
    entry = int(input())        # Get value from user
    if entry == 999:            # Did user provide the magic number?
        done = True             # If so, get out
    else:
        print(entry)            # If not, print it and continue
```

# The *while* statement

- Examples: sum of a set of nonnegative integers

```
entry = 0       # Ensure the loop is entered
sum = 0         # Initialize sum

#  Request input from the user
print("Enter numbers to sum, negative number ends list:")

while entry >= 0:           # A negative number exits the loop
    entry = int(input())  # Get the value
    if entry >= 0:          # Is number nonnegative?
        sum += entry        # Only add it if it is nonnegative
print("Sum =", sum)         # Display the sum
```

# The *while* statement

- Examples: find the associated factors of integers from 1 to 20.
  - Any problems?

```
#  List the factors of the integers 1...MAX
MAX = 20                                # MAX is 20
n = 1   # Start with 1
while n <= MAX:                         # Do not go past MAX
    factor = 1                          # 1 is a factor of any integer
    print(end=str(n) + ': ')            # Which integer are we examining?
    while factor <= n:                  # Factors are <= the number
        if n % factor == 0:             # Test to see if factor is a factor of n
            print(factor, end=' ')      # If so, display it
            factor += 1                 # Try the next number
    print()  # Move to next line for next n
    n += 1
```

# The *while* statement

- Examples: find the associated factors of integers from 1 to 20.
  - **Infinite loop**: executes its block of statements repeatedly until the user forces the program to quit
  - It prints:

# The *while* statement

- Examples: find the associated factors of integers from 1 to 20.
  - **Infinite loop**: executes its block of statements repeatedly until the user forces the program to quit

```
#  List the factors of the integers 1...MAX
MAX = 20                                # MAX is 20
n = 1   # Start with 1
while n <= MAX:                         # Do not go past MAX
    factor = 1                          # 1 is a factor of any integer
    print(end=str(n) + ': ')            # Which integer are we examining?
    while factor <= n:                  # Factors are <= the number
        if n % factor == 0:             # Test to see if factor is a factor of n
            print(factor, end=' ')      # If so, display it
            factor += 1                 # Try the next number
    print()  # Move to next line for next n
    n += 1
```

# The *while* statement

- Examples: find the associated factors of integers from 1 to 20.

```
#  List the factors of the integers 1...MAX
MAX = 20                                # MAX is 20
n = 1   # Start with 1
while n <= MAX:                         # Do not go past MAX
    factor = 1                          # 1 is a factor of any integer
    print(end=str(n) + ': ')            # Which integer are we examining?
    while factor <= n:                  # Factors are <= the number
        if n % factor == 0:             # Test to see if factor is a factor of n
            print(factor, end=' ')      # If so, display it
            factor += 1                 # Try the next number
    print()  # Move to next line for next n
    n += 1
```

Need to move this statement outside of the *if* statement

# Outline

- The *while* statement
- **The *for* statement**
- The *break* and *continue* statement
- Iteration examples

# The *for* statement

- The *for* statement is a definite loop, and it iterates over a sequence of values.

```
n = 1
while n <= 10:
    print(n)
    n += 1
```

```
for n in 1, 2, 3, 4, 5, 6, 7, 8, 9, 10:
    print(n)
```

# The *for* statement

- The *for* statement is a definite loop, and it iterates over a sequence of values.

```
for n in 1, 2, 3, 4, 5, 6, 7, 8, 9, 10:
    print(n)
```

```
for n in range(1, 11):
    print(n)
```

# The *for* statement

- Use the *range* function in the *for* statement.

range( *begin* , *end* , *step* )

- *begin* is the first value in the range; if omitted, the default value is 0.
- *end* is one past the last value in the range; the end value is always required and may not be omitted
- *step* is the amount to increment or decrement; if the step parameter is omitted, it defaults to 1 (counts up by ones).
- *begin*, *end*, and *step* are all integer expressions.

# The *for* statement

- Examples using the *range* function

`range(10)` → 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

`range(1, 10)` → 1, 2, 3, 4, 5, 6, 7, 8, 9

`range(1, 10, 2)` → 1, 3, 5, 7, 9

`range(10, 0, -1)` → 10, 9, 8, 7, 6, 5, 4, 3, 2, 1

`range(10, 0, -2)` → 10, 8, 6, 4, 2

`range(2, 11, 2)` → 2, 4, 6, 8, 10

# The *for* statement

- Nested loops: *while* and *for* loop blocks can contain other loops.
- Example: want to print a multiplication table

| × | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 2 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 |
| 3 | 3 | 6 | 9 | 12 | 15 | 18 | 21 | 24 | 27 | 30 |
| 4 | 4 | 8 | 12 | 16 | 20 | 24 | 28 | 32 | 36 | 40 |
| 5 | 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 |
| 6 | 6 | 12 | 18 | 24 | 30 | 36 | 42 | 48 | 54 | 60 |
| 7 | 7 | 14 | 21 | 28 | 35 | 42 | 49 | 56 | 63 | 70 |
| 8 | 8 | 16 | 24 | 32 | 40 | 48 | 56 | 64 | 72 | 80 |
| 9 | 9 | 18 | 27 | 36 | 45 | 54 | 63 | 72 | 81 | 90 |
| 10 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |

# The *for* statement

- Nested loops: *while* and *for* loop blocks can contain other loops.
- Example: want to print a multiplication table.
  - Codes:

```
#  Get the number of rows and columns in the table
size = int(input("Please enter the table size: "))
#  Print a size x size multiplication table
for row in range(1, size + 1):
    for column in range(1, size + 1):
        product = row*column                          # Compute product
        print('{0:4}'.format(product), end='')  # Display product
    print()                                           # Move cursor to next row
```

# The *for* statement

- Example: want to print a multiplication table.
  - Output:

```
Please enter the table size: 10
  1   2   3   4   5   6   7   8   9  10
  2   4   6   8  10  12  14  16  18  20
  3   6   9  12  15  18  21  24  27  30
  4   8  12  16  20  24  28  32  36  40
  5  10  15  20  25  30  35  40  45  50
  6  12  18  24  30  36  42  48  54  60
  7  14  21  28  35  42  49  56  63  70
  8  16  24  32  40  48  56  64  72  80
  9  18  27  36  45  54  63  72  81  90
 10  20  30  40  50  60  70  80  90 100
```

# Outline

- The *while* statement
- The *for* statement
- **The *break* and *continue* statement**
- Iteration examples

# The *break* and *continue* statements

- The *break* statement: exits a loop from the middle of its body, i.e., quits the loop before all the statements in its body.
- Example: sum a sequence of nonnegative numbers.

```
entry = 0       # Ensure the loop is entered
sum = 0         # Initialize sum

#  Request input from the user
print("Enter numbers to sum, negative number ends list:")

while True:                 # Loop forever?  Not really
    entry = int(input())    # Get the value
    if entry < 0:           # Is number negative number?
        break               # If so, exit the loop
    sum += entry            # Add entry to running sum
print("Sum =", sum)         # Display the sum
```

# The *break* and *continue* statements

- Using *break* versus not using *break*

```
while Condition 1 :
    Part A
    if Condition 2 :
        Part B
        break
    Part C
```

Eliminate the **break** statement

```
looping = True
while looping and Condition 1 :
    Part A
    if Condition 2 :
        Part B
        looping = False
    else:
        Part C
```

# The *break* and *continue* statements

- Using *break* in a *for* statement: counting the vowel

Codes:

```
word = input('Enter text (no X\'s, please): ')
vowel_count = 0
for c in word:
    if c == 'A' or c == 'a' or c == 'E' or c == 'e' \
       or c == 'I' or c == 'i' or c == 'O' or c == 'o':
        print(c, ', ', sep='', end='')  # Print the vowel
        vowel_count += 1                # Count the vowel
    elif c == 'X' or c =='x':
        break
print(' (', vowel_count, ' vowels)', sep='')
```

Output:

```
Enter text (no X's, please): Mary had a lixtle lamb.
a, a, a, i,  (4 vowels)
```

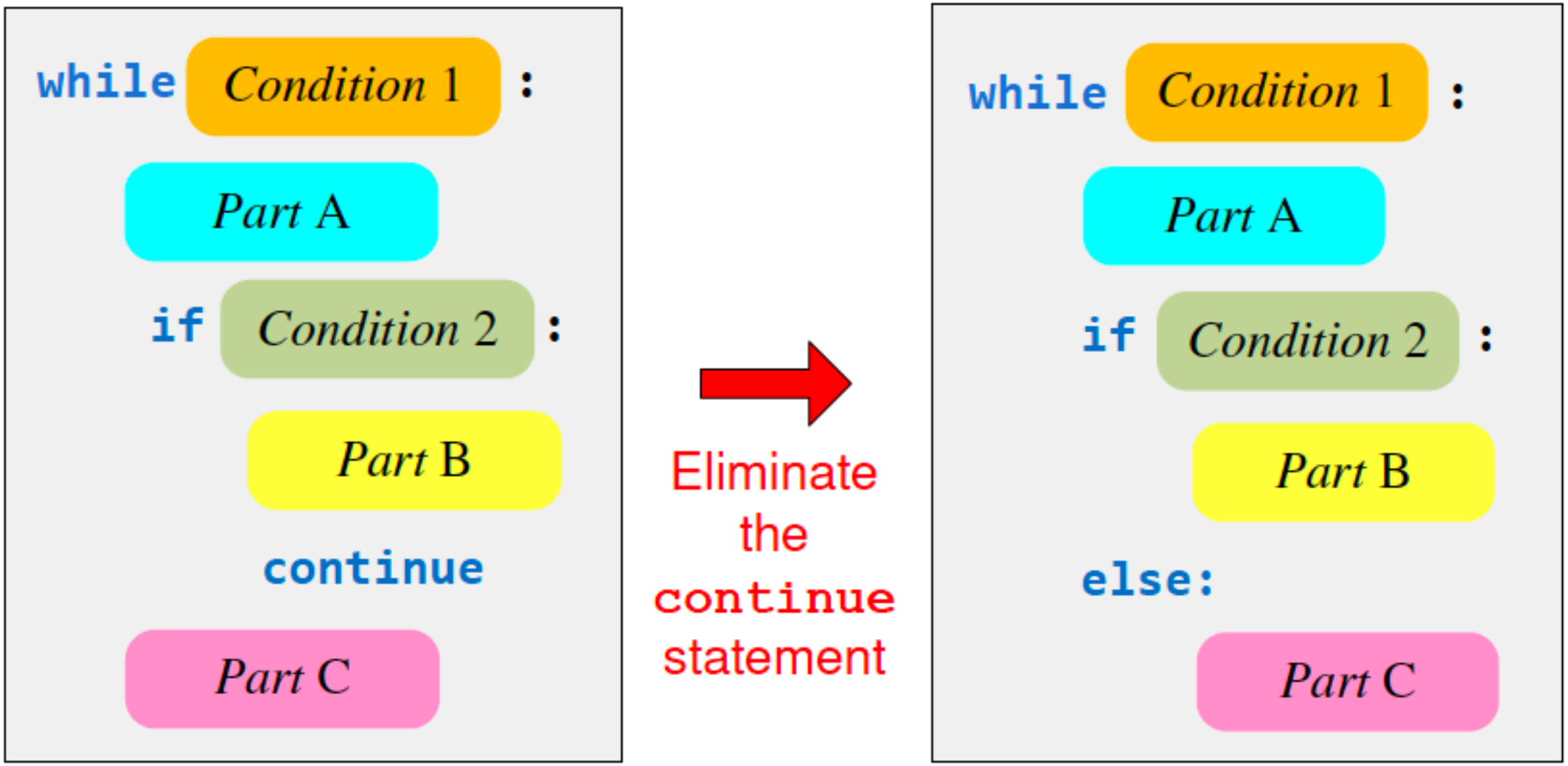
# The *break* and *continue* statements

- The *continue* statement: skips the rest of the body of the loop and immediately checks the loop's condition.

```
sum = 0
done = False
while not done:
    val = int(input("Enter positive integer (999 quits):"))
    if val < 0:
        print("Negative value",  val,  "ignored")
        continue  # Skip rest of body for this iteration
    if val != 999:
        print("Tallying", val)
        sum += val
    else:
        done = (val == 999)   # 999 entry exits loop
print("sum =", sum)
```

# The *break* and *continue* statements

- Using *continue* versus not using *continue*



# Outline

- The *while* statement
- The *for* statement
- The *break* and *continue* statement
- **Iteration examples**

# *Iteration examples*

- Drawing a tree


A tree is 5 levels tall.

# *Iteration examples*

- Drawing a tree (using *while* loop)

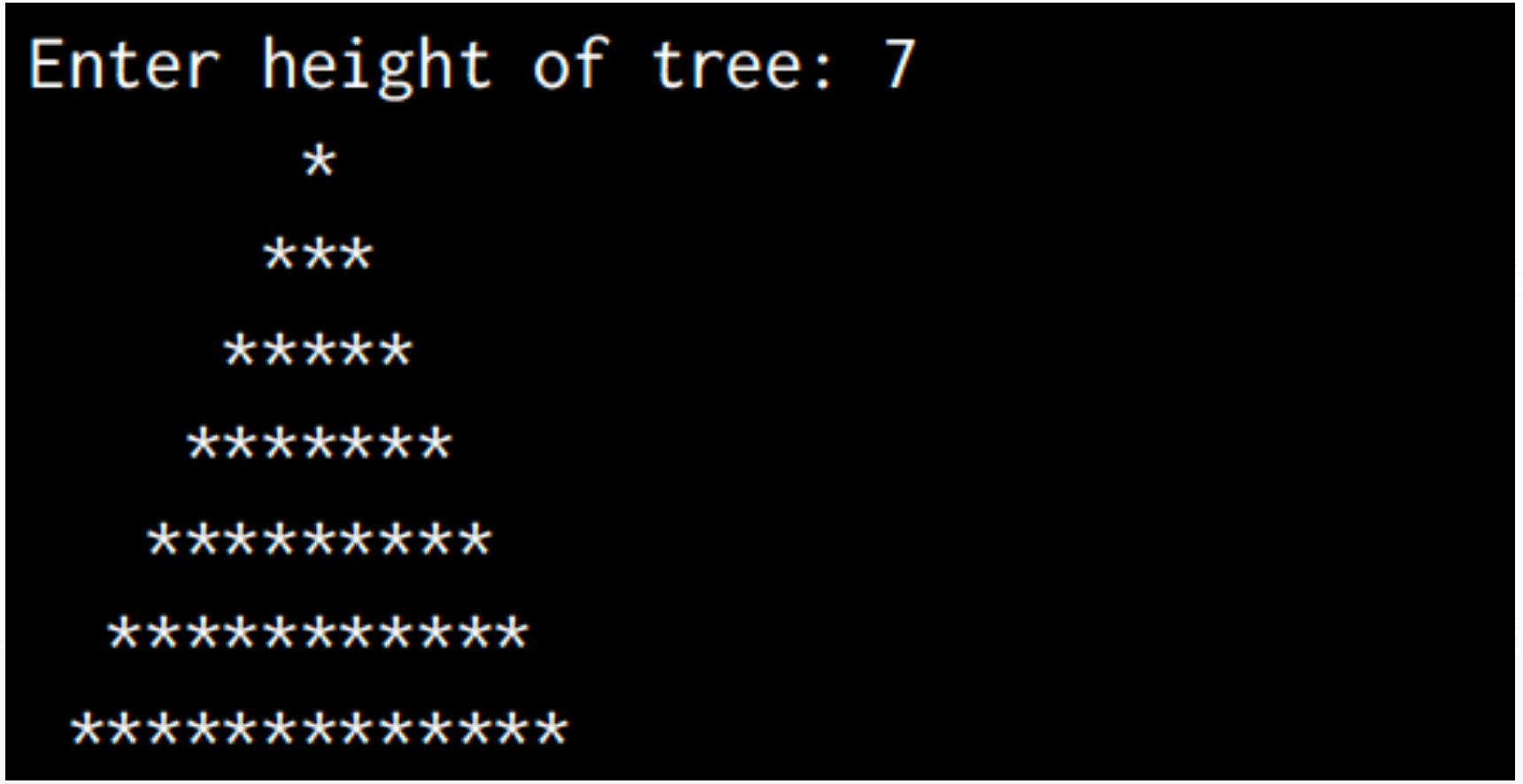
Codes

```
#  Get tree height from user
height = int(input('Enter height of tree: '))

#  Draw one row for every unit of height
row = 0
while row < height:
    # Print leading spaces; as row gets bigger, the number of
    # leading spaces gets smaller
    count = 0
    while count < height - row:
        print(end=' ')
        count += 1

    # Print out stars, twice the current row plus one:
    #   1. number of stars on left side of tree
    #      = current row value
    #   2. exactly one star in the center of tree
    #   3. number of stars on right side of tree
    #      = current row value
    count = 0
    while count < 2*row + 1:
        print(end='*')
        count += 1
    # Move cursor down to next line
    print()
    row += 1   # Consider next row
```

# *Iteration examples*

- Drawing a tree (using *while* loop)

Output



# *Iteration examples*

- Drawing a tree (using *for* loop)

```
#  Get tree height from user
height = int(input('Enter height of tree: '))

#  Draw one row for every unit of height
for row in range(height):
    # Print leading spaces; as row gets bigger, the number of
    # leading spaces gets smaller
    for count in range(height - row):
        print(end=' ')

    # Print out stars, twice the current row plus one:
    #   1. number of stars on left side of tree
    #       = current row value
    #   2. exactly one star in the center of tree
    #   3. number of stars on right side of tree
    #       = current row value
    for count in range(2*row + 1):
        print(end='*')
    # Move cursor down to next line
    print()
```

# *Iteration examples*

- Printing prime numbers (using *while* loop)

Codes

```
max_value = int(input('Display primes up to what value? '))
value = 2  # Smallest prime number
while value <= max_value:
    # See if value is prime
    is_prime = True  # Provisionally, value is prime
    # Try all possible factors from 2 to value - 1
    trial_factor = 2
    while trial_factor < value:
        if value % trial_factor == 0:
            is_prime = False    # Found a factor
            break               # No need to continue; it is NOT prime
        trial_factor += 1       # Try the next potential factor
    if is_prime:
        print(value, end= ' ')  # Display the prime number
    value += 1                  # Try the next potential prime number
print()  # Move cursor down to next line
```

# *Iteration examples*

- Printing prime numbers (using *while* loop)

Output

```
Display primes up to what value? 90
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89
```

# *Iteration examples*

- Printing prime numbers (using *for* loop)

Codes

```
max_value = int(input('Display primes up to what value? '))
#  Try values from 2 (smallest prime number) to max_value
for value in range(2, max_value + 1):
    # See if value is prime
    is_prime = True  # Provisionally, value is prime
    # Try all possible factors from 2 to value - 1
    for trial_factor in range(2, value):
        if value % trial_factor == 0:
            is_prime = False    # Found a factor
            break               # No need to continue; it is NOT prime
    if is_prime:
        print(value, end= ' ')  # Display the prime number
print()  # Move cursor down to next line
```

# Readings

- Think Python, chapter 7

# References


- *Think Python: How to Think Like a Computer Scientist*, 2nd edition, 2015, by Allen Downey
- *Python Cookbook*, 3rd edition, by David Beazley and Brian K. Jones, 2013
- *Python for Data Analysis*, 2nd edition, by Wes McKinney, 2018
- *Fundamentals of Python Programming,* by Richard L. Halterman