

Course	
Term	
Week	
Date	
Chapter. Topic	4. Repetition Structures

Random number generation

Continue, Break

Range function and For Loops

Extending for and while loops

Siva R Jasthi

Computer Science and Cybersecurity

Metropolitan State University

1

Random Number Generation

- Using “random” library
- Typically used to make random choices/selections

Outline

2

WHILE loops

- While loops (indefinite loops)
- https://www.w3schools.com/python/python_while_loops.asp

3

Break

- for breaking out of the loop
- Use to exit the loop based on a condition

4

Continue

- For starting the execution at the top of the loop
- For ignoring the rest of the code block in the current thread

5

WHILE loops with Sentinels (while True)

- While loops (Sentinel loop) – Use a value to terminate the loop
- https://www.w3schools.com/python/python_while_loops.asp

Dictionary

Definitions from [Oxford Languages](#) · [Learn more](#)



sen·ti·nel

/'sentənəl,'sentn(ə)l/

noun

a soldier or guard whose job is to stand and keep watch.

6

Range function

- generate a list of numbers
- used in for loops
- One or Two or Three arguments

7

For loop

- definite – for a fixed number of times.
- known as “count-controlled” loop
- https://www.w3schools.com/python/python_for_loops.asp

8

Nested Loops

- Loops inside loops (indented)
- for and while loops can be mixed
- Nesting can be deep (but not good idea)

9

Extending the while loops with else

If a loop completes all its iterations normally, then the code inside the “else” block will be executed.

10

Extending the for loops with else

If a loop completes all its iterations normally, then the code inside the “else” block will be executed.

1

Random Number Generation

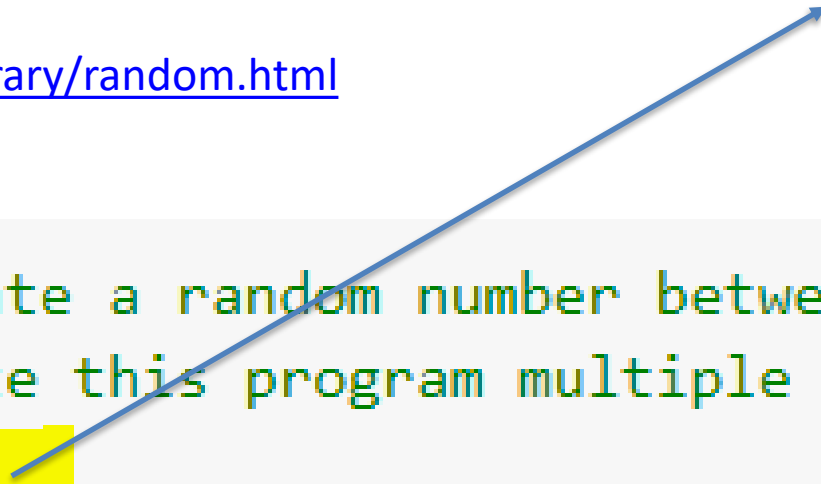
- Using “random” library
- Typically used to make random choices/selections

Generating random numbers in python

- Generate a random number
- Generate a random number between two numbers

Pay attention to the “import” statement.
This is a python “module”

<https://docs.python.org/3/library/random.html>



```

1 #@title Generate a random number between A and B
2 # Note: Execute this program multiple times
3 import random
4 A = 10
5 B = 100
6 random_x = random.randint(A,B)
7 print("Random Number between", A, "and", B, "is: ", random_x)
  
```


“random” library

- Some functions are included in the basic python.

For example: `input()`, `print()`

- Some advanced functions are included in separate packages.

packages = libraries = modules

- Difference between package, library, module, framework

<https://learnpython.com/blog/python-modules-packages-libraries-frameworks>



<https://pypi.org/>

WHILE loops

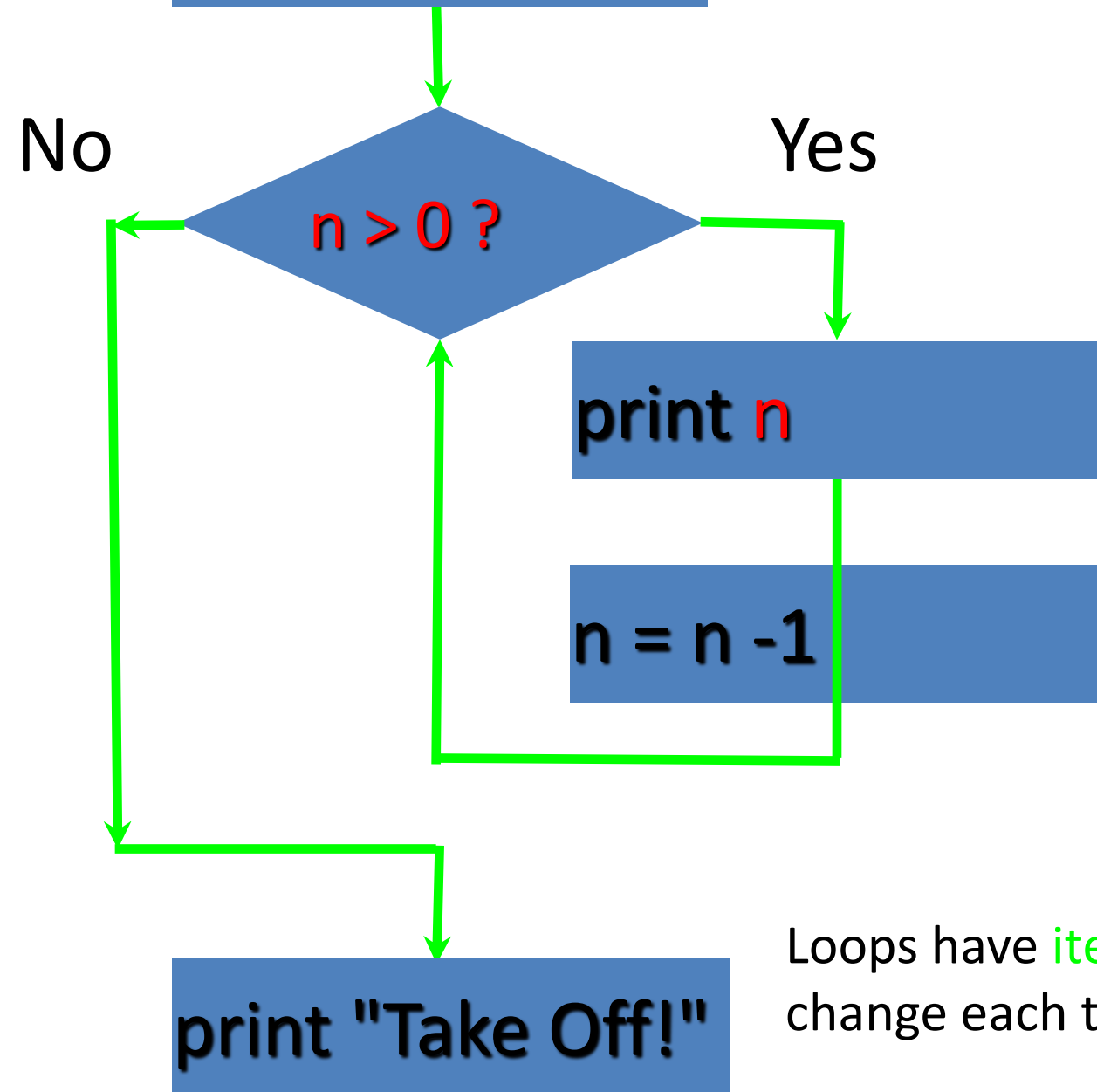
2

- While loops (indefinite loops)
- While loops (Sentinel loop) – Use a value to terminate the loop
- https://www.w3schools.com/python/python_while_loops.asp

Program:

```
n = 10
while n > 0 :
    print(n)
    n = n-1
print("Take Off!")
print(n)
```

While loop



Program:

```

n = 5
while (n > 0) :
    print(n)
    n = n-1
print("Take Off!")
print(n)
  
```

Output:

5
4
3
2
1
Take Off!
0

Loops have **iteration variables** that change each time through a loop.

Let us visualize this code on Python Tutor

The link is [here](#).

Python 3.6
([known limitations](#))

```
1 n = 100
2 while n > 0 :
→ 3     print(n)
→ 4     n = n-1
5 print("Take Off!")
6 print(n)
```

[Edit this code](#)

Print output (drag lower)

```
100
99
98
97
```

Frames

Global frame

n 97

< First

< Prev

Next >

Last >>

Step 13 of 304

Sometimes the loop code may not execute..

Watch out the loop variable.

Is the condition false when you started the loop?

Then code in the loop will never execute!



What gets printed?

Program:

```
n = 5
while n < 0 :
    print(n)
    n = n-1
print(n)
```

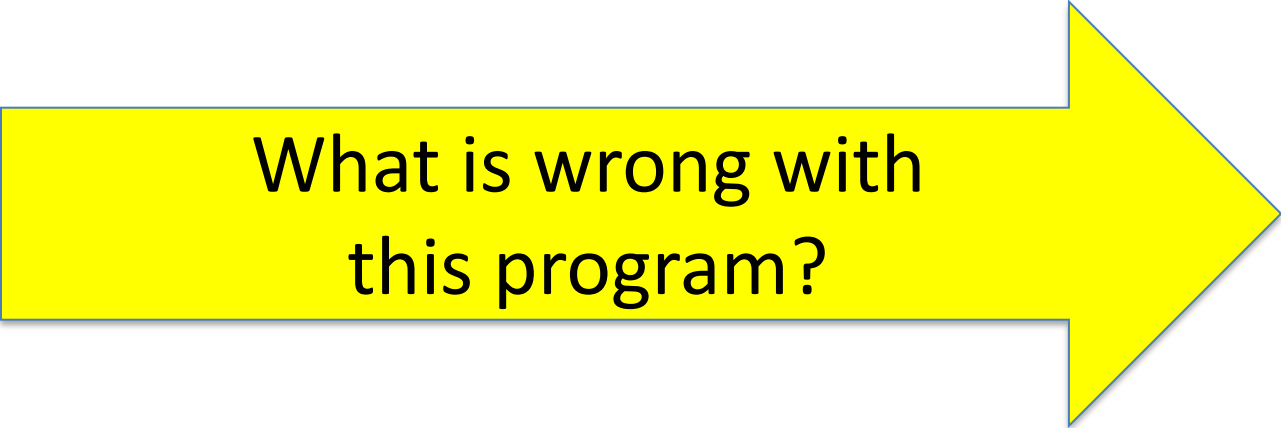
The loop need to stop at some point...

Watch out the loop variable.

Will the condition be true at some point?

If not, we will have a problem.

The loop runs forever!



What is wrong with
this program?

Program:

```
n = 5
while n > 0 :
    print(n)
print("Take Off!")
print(n)
```

The loop need to stop at some point...

Watch out the loop variable.

Will the condition be true at some point?

If not, we will have a problem.

The loop runs forever!

What is wrong with
this program?

Program:

```
n = 5
while n > 0 :
    print(n)
    n = n+1
print("Take Off!")
print(n)
```

What to watch in 'while' loop?

1. Condition: Will it ever become FALSE?
2. Body: How are we converging towards FAILURE?

Typical steps while dealing with WHILE loops

1. Initialization: Declare and initialize variables before the loop starts.
2. Printing: Do it after the loop ends

3

Break

- for breaking out of the loop
- Use to exit the loop based on a condition

break - when is it used?

Consider this program. It is printing all numbers from 100 to 0.

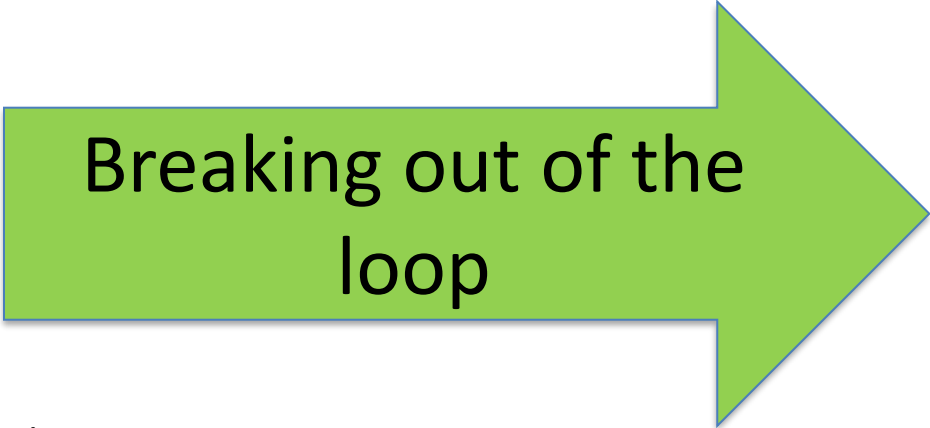
Let us say I do not like the number 13.

Once I come across 13, I want to stop the loop execution.

We can use “break” to break out of the loop.

Program:

```
n = 100
while n > 0 :
    print(n)
    n = n-1
print("Take Off!")
print(n)
```



**Breaking out of the
loop**

```
n = 20
while n > 0 :
    if (n == 13) :
        break
    else:
        print(n)
        n = n-1
print("Take Off!")

if (n != 13):
    print(n)
```

4

Continue

- For starting the execution at the top of the loop
- For ignoring the rest of the code block in the current thread

continue - when is it used?

Consider this program. It is printing all numbers from 100 to 0.

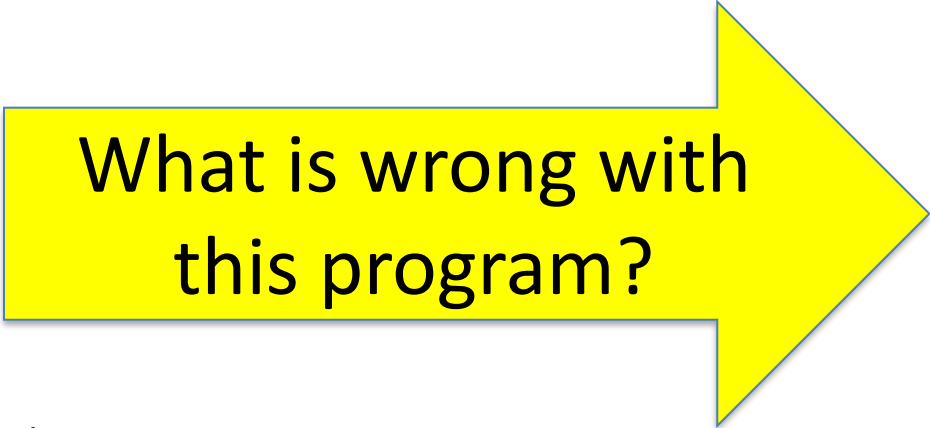
Let us say I don't like the number 13.

We need to throw in if condition to check for it.

And "continue" the loop after skipping if the number is 13.

Program:

```
n = 100
while n > 0 :
    print(n)
    n = n-1
print("Take Off!")
print(n)
```



**What is wrong with
this program?**

```
n = 15
while n > 0 :
    if (n == 13):
        continue
    else:
        print(n)
        n = n-1
print("Take Off!")
print(n)
```

continue - when is it used? (contd.)

Consider this program. It is printing all numbers from 100 to 0.

Let us say I don't like the number 13.

We need to throw in if condition to check for it.

And "continue" the loop after skipping if the number is 13.

Program:

```
n = 100
while n > 0 :
    print(n)
    n = n-1
print("Take Off!")
print(n)
```



With correction

```
n = 15
while n > 0 :
    if (n == 13):
        n = n-1
        continue
    else:
        print(n)
        n = n-1
print("Take Off!")
print(n)
```

5

WHILE loops with Sentinels

- While loops (Sentinel loop) – Use a value to terminate the loop
- https://www.w3schools.com/python/python_while_loops.asp

sentinel - when is it used?

Sometimes, you may want to break out of the loop when you come across a special value.

That special value is called “sentinel”.

Usually used to exit a loop when user indicates to end a loop.

Program:

```
n = 100
```

```
while True:
```

```
    response = input('Would you like to continue? (yes/no): `')
```

```
    if response == 'no':
```

```
        break
```

```
print(n)
```

```
n = n - 1
```

(start : stop : step)

(0 : stop : 1)

Range function

- to generate a list of numbers
- used in for loops
- One argument. Two arguments. Three Arguments
 - 0 (zero) to N
 - M to N
 - M to N in increments of "S"

range function

STOP
START, STOP
START, STOP, STEP

	Syntax	Output
Single Argument	<code>range(j)</code> Ex: <code>range(10)</code>	$0, 1, 2, 3, 4, \dots, j-1$ <code>0, 1, 2, 3, 4, 5, 6, 7, 8, 9</code>
Double Argument	<code>range(i, j)</code> Ex: <code>range(1, 10)</code>	$i, i+1, i+2, i+3, \dots, j-1$ <code>1, 2, 3, 4, 5, 6, 7, 8, 9</code>
Triple Argument	<code>range(i, j, k)</code> Ex: <code>range(1, 10, 2)</code>	$i, i+k, i+2k, i+3k, \dots, j-1$ <code>1, 3, 5, 7, 9</code>

range() function

range() function starts at 0 (by default)

range() function stops at the second number (it doesn't include it in the range)

range() function also takes an increment number as 3rd parameter)

```
for x in range(6):  
    print(x)
```

0, 1, 2, 3, 4, 5

```
for x in range(2, 6):  
    print(x)
```

2, 3, 4, 5

```
for x in range(2, 30, 3):  
    print(x)
```

2, 5, 8, 11, 14, 17, 20, 23, 26, 29

The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

7

For loop

- definite – for a fixed number of times.
- known as “count-controlled” loop
- https://www.w3schools.com/python/python_for_loops.asp

What is a “sequence”?

Here are some example of a sequence

`range(5) → [0, 1, 2, 3, 4]`

`range(4, 14, 3) → [4, 7, 10, 13]`

Strings are a sequence of characters → `“python” → [“p”, “y”, “t”, “h”, “o”, “n”]`

List of numbers → `[5, 6, 7, 7, 8, 6, 5]`

List of characters → `[‘a’, ‘b’, ‘c’, ‘d’, ‘e’]`

List of strings → `[‘hitagna’, ‘bhargav’, ‘siva’]`

List of mixed items → `[‘john’, 30, ‘A’, 98]`

Tuple = list of items that doesn’t change = `(‘mon’, ‘tue’, ‘wed’, ‘thu’, ‘fri’, ‘sat’, ‘sun’)`

Set of items (no duplicates) = `{3, 5, 7}`

Dictionary = `{‘a’: ‘apple’, ‘b’: ‘berry’, ‘c’: ‘cherry’}`

What does it mean by “iterating over” a sequence?

`range(4, 14, 3) → [4, 7, 10, 13]`

- Iterate over a sequence
- Visit each and every element in the sequence
- Loop over a sequence

for loops: are definite loops

Consider these collections:

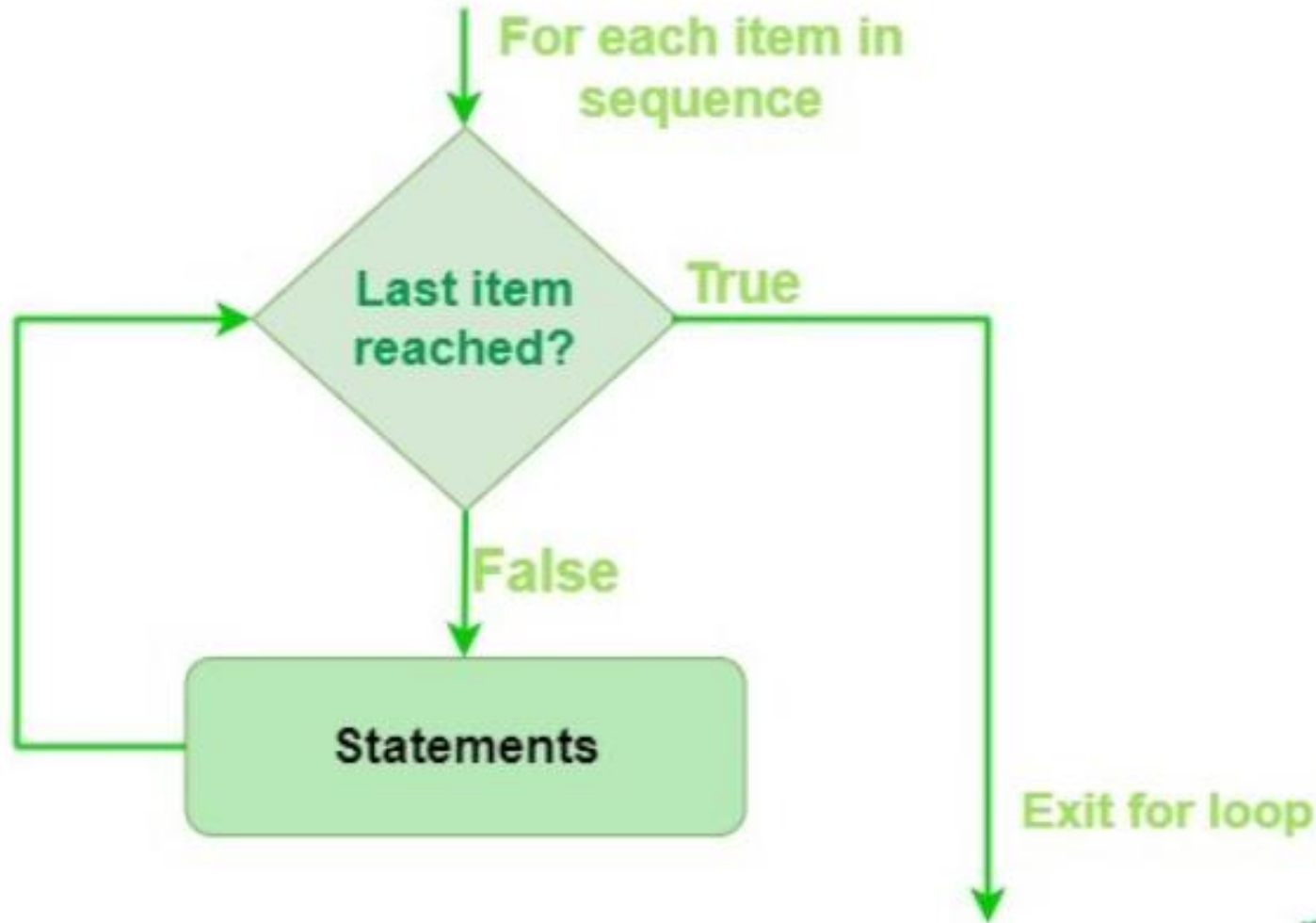
- Items in a list.
- Lines in a file.
- Characters in a string.
- Numbers in a set.

We frequently iterate over these items for a fixed number of times.

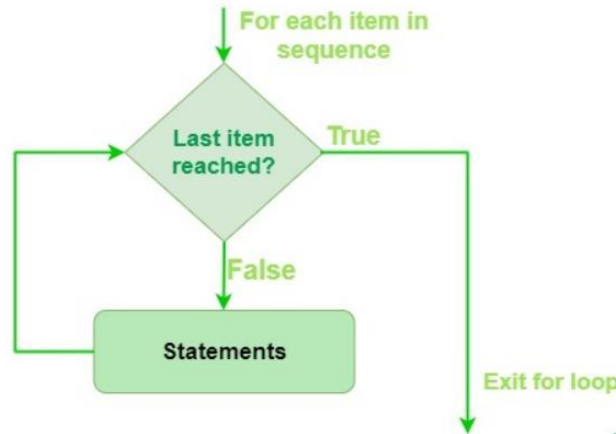
(Number of times = Number of items in the collection)

These loops are called "**definite loops**" because they execute an exact number of times

for loops: are definite loops



for loops: are definite loops



Definite loops (for loops) have explicit **iteration variables** that change each time through a loop.

These **iteration variables** move through the sequence or set.

```
students = ["abe", "barb", "chris", "a  
be", "dan", "chris", "ellie"]
```

```
for x in students:  
    print(x)
```


for loops: are definite loops

As discussed in while loops, you can use “continue” and “break” statements in the for loops as well.

Break: to exit out of the loop

Continue: to re-start the execution at the beginning of the loop.

8

Nested Loops

- Loops inside loops (indented)
- for and while loops can be mixed
- Nesting can be deep (but not good idea)

Nested for loops

You can also have a for loop inside a for loop.

Assume the outer for loop has a range of 10.
And the inner for loop has a range of 9.

Then the loop statements will run $10 * 9 = 90$ times.

Nested Loops: loops inside loops

Nested Loop:

A loop that is inside another loop is called a nested loop.

```
1 hours = 0
2 minutes = 0
3 seconds = 0
4 while hours < 24:
5     while minutes < 60:
6         while seconds < 60:
7             print(hours, ': ', minutes, ': ', seconds)
8             seconds = seconds + 1
9             minutes = minutes + 1
10            seconds = 0
11        hours = hours + 1
12        minutes = 0
```

Nested Loops: Multiplication Table

Nested Loop:

A loop that is inside another loop is called a nested loop.

```
1 for x in range(1,10):  
2     for y in range(1,10):  
3         print(x, "*", y, "=", (x*y))  
4         print()
```

Loops: Summary

1. While loops
2. Break
3. Continue
4. Range
5. For Loops
6. Nested Loops

- A. Condition-Controlled Loop
- B. Count Controlled Loop
- C. Sentinel Loop

- I. Infinite Loops
- II. Finite Loops

Outline for today

9

Extended for loops

10

Extended while loops

Extended for loop

The **else** block in a while or for loop is used

- to execute a block of code when the loop has completed **all its iterations**,

i.e., when the loop condition becomes false

OR

when there are no more items to iterate over.

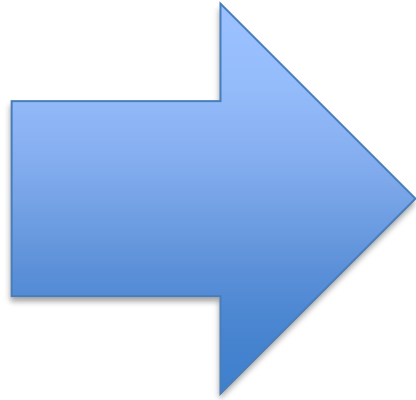
else block is not executed if there is a 'break' in the loop.

else block is executed only if all the iterations (loops) are completed.

<https://www.geeksforgeeks.org/using-else-conditional-statement-with-for-loop-in-python/>

Extended while loop

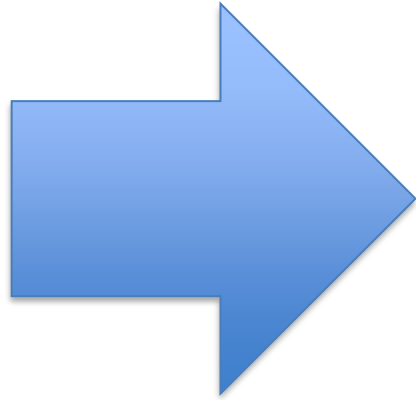
```
n = 5  
while n > 0 :  
    print(n)  
    n = n-1  
else:  
    print(0)
```



```
n = 5  
while n > 0 :  
    print(n)  
    n = n-1  
  
print(0)
```

Extended for loop

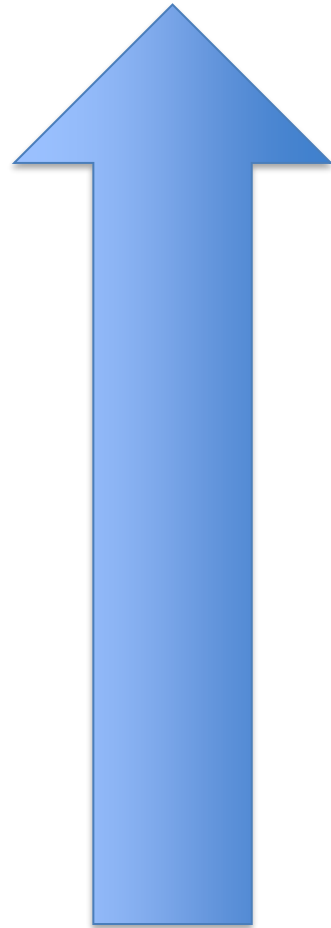
```
for x in range(1,6):  
    print(x)  
else:  
    print(0)
```



```
for x in range(1,6):  
    print(x)  
print(0)
```

Extended for loop

```
for x in range(1,6):  
    print(x)  
    break  
else:  
    print(0)
```



```
for x in range(1,6):  
    print(x)  
print(0)
```

If a loop comes across a “break”, that implies that it did not run all its iterations.
In such case, “else” will not execute

What is the use of 'else' then?

To indicate that some logic is associated with while or for block.

If you keep it outside the block, it may be assumed that it has nothing to do with the iteration.

Other than improving the readability of 'iteration', I do not see any other purpose of 'else' block.

Loops: Summary

1. While loops
2. Break
3. Continue
4. Range
5. For Loops
6. Nested Loops

- A. Condition-Controlled Loop
- B. Count Controlled Loop
- C. Sentinel Loop

- I. Infinite Loops
- II. Finite Loops

- I. Extending while loops
- II. Extending for loop

Thank You.

PYTHON PROGRAMMING

by SIVA JASTHI