



CMSC 105 Elementary Programming

Acknowledgement: These slides are adapted from slides provided with "Introduction to Programming Using Python, Liang (Pearson 2013)" and slides shared by Dr. Jory Denny

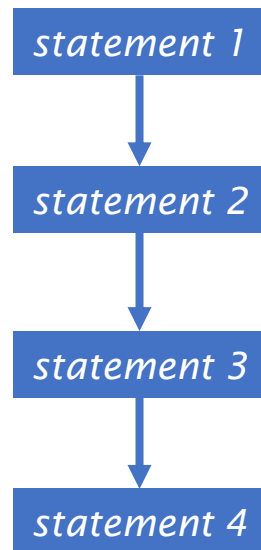
Outline

The **While** Loops

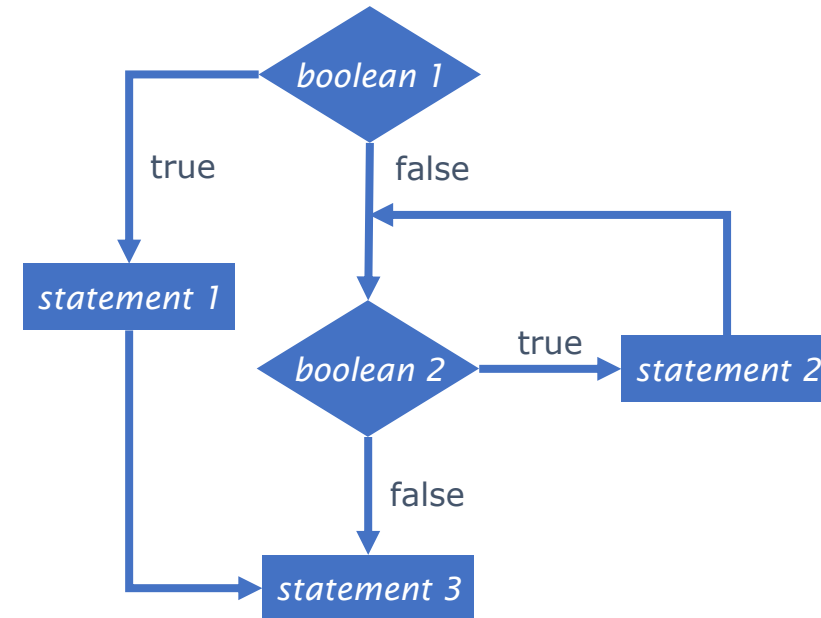
The **For** Loops

Control Flow

- Control flow.
 - Sequence of statements that are actually executed in a program.
 - Conditionals and loops: enable us to choreograph control flow.



straight-line control
flow



control flow with conditionals
and loops

Motivations

- Suppose that you need to print a string (e.g., "Welcome to Python!") a hundred times. It would be tedious to have to write the following statement a hundred times:

```
print("Welcome to Python!")
```

- So, how do you solve this problem?
- How about altering our guessing game program to allow 20 tries?

A loop can be used to tell a program to execute statements repeatedly!

Opening Problem

```
print("Welcome to Python!")  
print("Welcome to Python!")  
print("Welcome to Python!")  
print("Welcome to Python!")  
print("Welcome to Python!")  
print("Welcome to Python!")  
  
...  
print("Welcome to Python!")  
print("Welcome to Python!")
```

} 100
times

The While Loops



Image from <http://www.quickmeme.com/meme/3rgii1/page/1/>

Introducing while Loops

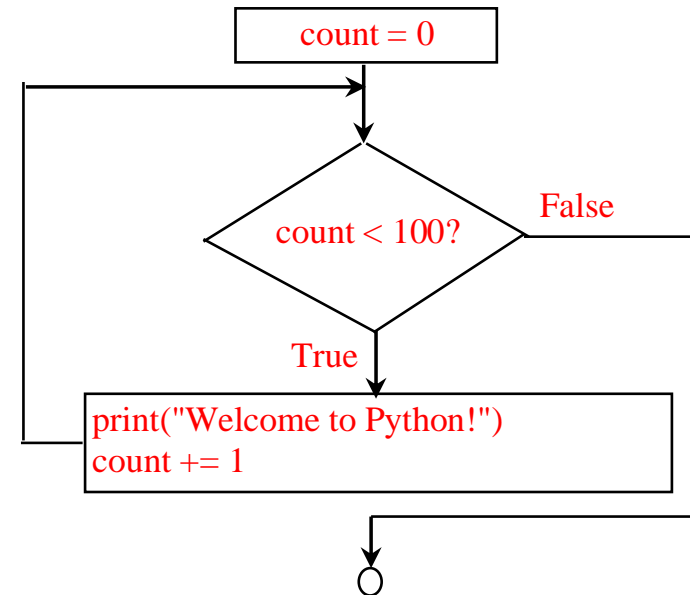
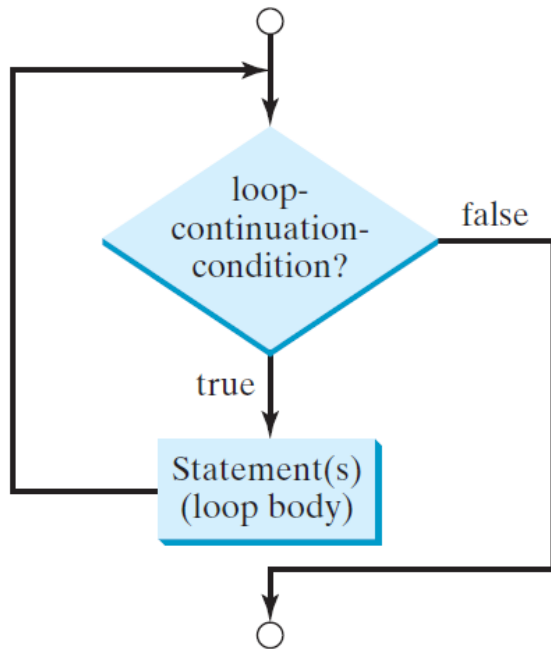
```
1. count = 0
2. while count < 100:
3.     print("Welcome to Python")
4.     count += 1
```

while Loop Flow Chart

```
1. while loop-continuation-condition:  
2.     # loop-body  
3.     Statement(s)
```

```
1. count = 0  
2. while count < 100:  
3.     print("Welcome to Python")  
4.     count += 1
```

A while loop executes statements repeatedly as long as a condition remains true



Tracing While loops

```
1. count = 0
2. while count < 2:
3.     print("Welcome to Python")
4.     count += 1
```

Memory

Output

Tracing While loops

```
1. count = 0
2. while count < 2:
3.     print("Welcome to Python")
4.     count += 1
```

Initialize Count



Memory
count: 0

Output

Tracing While loops

```
1. count = 0
2. while count < 2:
3.     print("Welcome to Python")
4.     count += 1
```

Count < 2 is true

Memory

count: 0

Output

Tracing While loops

```
1. count = 0
2. while count < 2:
3.     print("Welcome to Python")
4.     count += 1
```

Output

Memory

count: 0

Output

Welcome to Python

Tracing While loops

```
1. count = 0
2. while count < 2:
3.     print("Welcome to Python")
4.     count += 1
```

Increment count



Memory

count: 0 1


Output

Welcome to Python

Tracing While loops

```
1. count = 0
2. while count < 2:
3.     print("Welcome to Python")
4.     count += 1
```

Count < 2 is true



Memory

count: 0 1

Output

Welcome to Python

Tracing While loops

```
1. count = 0
2. while count < 2:
3.     print("Welcome to Python")
4.     count += 1
```

Output

Memory

count: 0 1

Output

Welcome to Python
Welcome to Python

Tracing While loops

```
1. count = 0
2. while count < 2:
3.     print("Welcome to Python")
4.     count += 1
```

Increment count



Memory

count: 0	1	2
----------	---	---

Output

Welcome to Python
Welcome to Python

Tracing While loops

```
1. count = 0
2. while count < 2:
3.     print("Welcome to Python")
4.     count += 1
```

Count < 2 is false

Memory

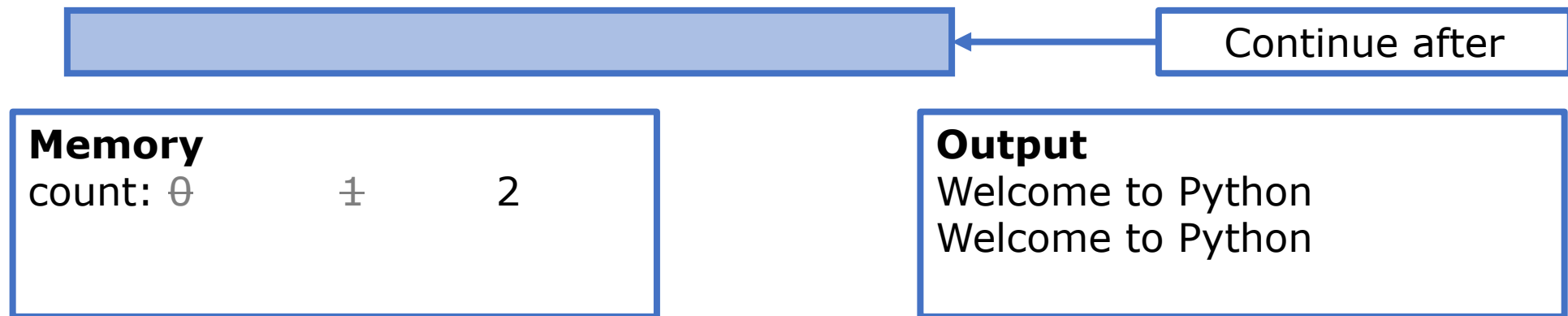
count: 0	1	2
----------	---	---

Output

Welcome to Python
Welcome to Python

Tracing While loops

```
1. count = 0
2. while count < 2:
3.     print("Welcome to Python")
4.     count += 1
```



Question

- What is wrong with the following code?
- What happens?
- Fix it and explain what the code outputs

```
1. i, N = 0, 10000
2. while i <= N:
3.     print(i)
4. i = i + 5
```

Exercise

- Using while loops, write a program that runs 10 times and prints the sum of numbers from 1 to 10

Ending a Loop with a Sentinel Value

Often the number of times a loop is executed is not predetermined. You may use an input value to signify the end of the loop. Such a value is known as a *sentinel value*.

Write a program that reads and calculates the sum of an unspecified number of integers. The input 0 signifies the end of the input.

SentinelValue

Caution

- Don't use floating-point values for equality checking in a loop control. Since floating-point values are approximations for some values, using them could result in imprecise counter values and inaccurate results. Consider the following code for computing $1 + 0.9 + 0.8 + \dots + 0.1$:

```
1.item, sum = 1, 0
2.while item != 0: # No guarantee item will be 0
3.    sum += item
4.    item -= 0.1
5.print(sum)
```

The For Loop

```
#include <stdio.h>
int main(void)
{
    int count;

    for (count = 1; count <= 500; count++)
        printf("I will not throw paper airplanes in class.");

    return 0;
}
```

AMEND 10-3

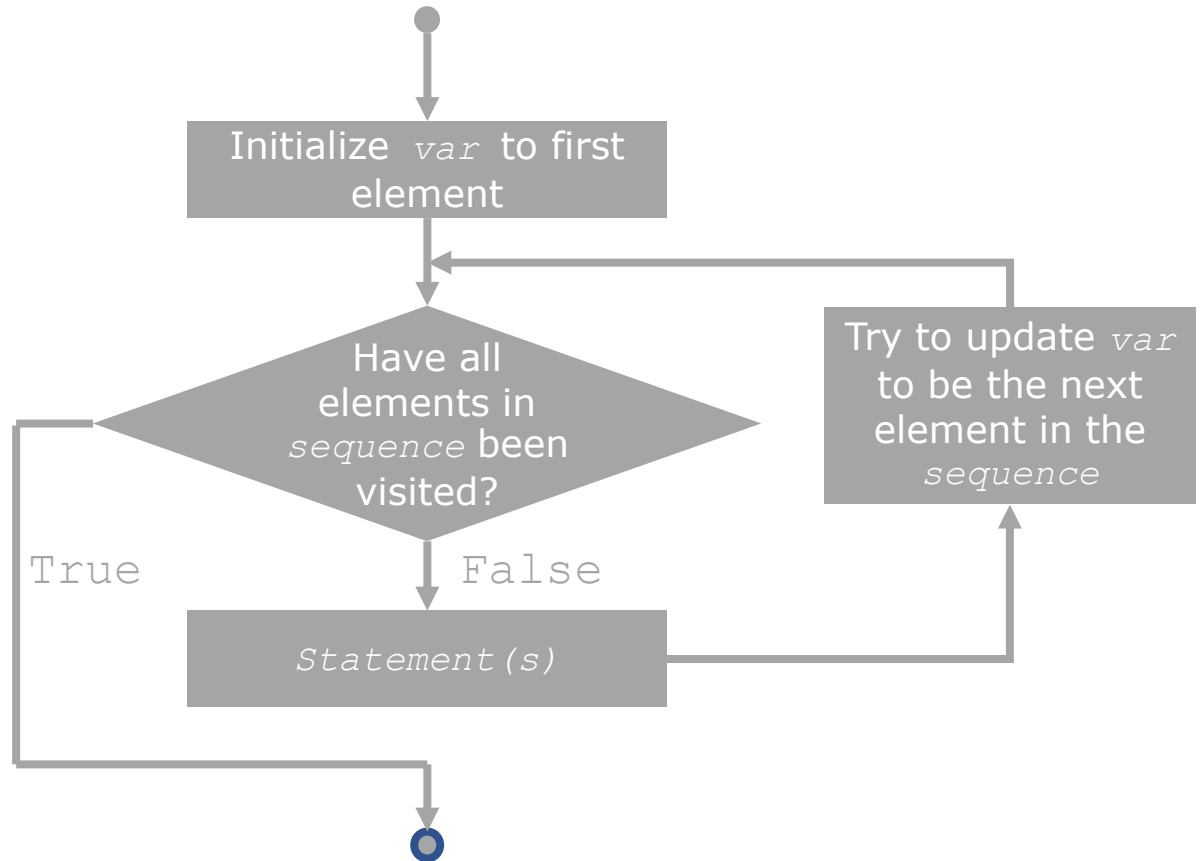


for Loops

```
1.  for var in sequence:  
2.      # loop body  
3.      Statement(s)
```

Example

```
1.  for x in range(0, 100):  
2.      print("Welcome to Python!")
```



for Loops

```
i = initialValue # Initialize loop-control variable
while i < endValue:
    # Loop body
    ...
    i++ # Adjust loop-control variable
```

```
for i in range(initialValue, endValue):
    # Loop body
```

range(a, b)

```
>>> for v in range(4, 8):  
...     print(v)  
...  
4  
5  
6  
7  
>>>
```

range(b)

```
>>> for i in range(4):  
...     print(i)  
...  
0  
1  
2  
3  
>>>
```

range(a, b, step)

```
>>> for v in range(3, 9, 2):  
...     print(v)  
...  
3  
5  
7  
>>>
```

range(a, b, step)

```
>>> for v in range(5, 1, -1):  
...     print(v)  
...  
5  
4  
3  
2  
>>>
```

Tracing For loops


```
1. for x in range(0, 2):  
2.     print("Welcome to Python!")
```

Memory

Output

Tracing For loops

```
1. for x in range(0, 2):  
2.     print("Welcome to Python!")
```



Initialize x

Memory

x: 0

**Note* range(0, 2) is [0, 1]*

Output

Tracing For loops

```
1. for x in range(0, 2):  
2.     print("Welcome to Python!")
```

Have all elements been visited?
No

Memory

x: 0

**Note* range(0, 2) is [0, 1]*

Output

Tracing For loops

1. `for x in range(0, 2):`

2. `print("Welcome to Python!")`

Output

Memory

x: 0

**Note* range(0, 2) is [0, 1]*

Output

Welcome to Python!

Tracing For loops

```
1. for x in range(0, 2):  
2.     print("Welcome to Python!")
```

Try to set x to next
element of
sequence

Memory

x: 0 1

**Note* range(0, 2) is [0, 1]*

Output

Welcome to Python!

Tracing For loops

```
1. for x in range(0, 2):  
2.     print("Welcome to Python!")
```

Have all elements been visited?
No

Memory

x: 0 1

**Note* range(0, 2) is [0, 1]*

Output

Welcome to Python!

Tracing For loops

1. `for x in range(0, 2):`

2. `print("Welcome to Python!")`

Output

Memory

x: 0 1

**Note* range(0, 2) is [0, 1]*


Output

Welcome to Python!
Welcome to Python!

Tracing For loops

```
1. for x in range(0, 2):  
2.     print("Welcome to Python!")
```

Try to set x to next
element of
sequence



Memory

x: 0 1

**Note* range(0, 2) is [0, 1]*

Output

Welcome to Python!
Welcome to Python!

Tracing For loops

```
1. for x in range(0, 2):  
2.     print("Welcome to Python!")
```

Have all elements been visited?
Yes

Memory

x: 0 1

**Note* range(0, 2) is [0, 1]*

Output

Welcome to Python!
Welcome to Python!

Tracing For loops

```
1. for x in range(0, 2):  
2.     print("Welcome to Python!")
```



Memory

x: 0 1

**Note* range(0, 2) is [0, 1]*

Output

Welcome to Python!
Welcome to Python!

Compare For Loops to while loop

```
1. count = 0
2. while count < 100:
3.     print("Welcome to Python")
4.     count += 1
```

```
1. for x in range(1, 100):
2.     print("Welcome to Python!")
```

Note, each has their own use.
For loops are a special case in which each element of a sequence is visited. In this case (and only this case) are for-loops appropriate in Python.

Other Control flow
statements

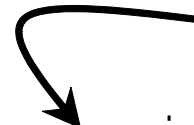
break

```
sum = 0  
number = 0
```

```
while number < 20:  
    number += 1  
    sum += number  
    if sum >= 100:
```

```
        break
```

Break out of
the loop



```
    print("The number is ", number)  
    print("The sum is ", sum)
```


continue

```
sum = 0
number = 0

while number < 20:
    number += 1
    if number == 10 or number == 11:
        continue
    sum += number

print("The sum is ", sum)
```

Jump to the
end of the
iteration



Example

Using for loop, write a program that:

- Runs 10 times (from 1 to 10)
- Computes square of numbers from 1 to 10 and prints them
- If the square value is greater than or equal to 80, then exit the for loop

Control Flow Summary

- Control flow
 - Sequence of statements that are actually executed in a program.
 - Conditionals and loops: enable us to choreograph the control flow.

Control Flow	Description	Examples
Straight-line programs	All statements are executed in the order given	
Conditionals	Certain statements are executed depending on the values of certain variables	<code>if; if-else</code>
Loops	Certain statements are executed repeatedly until certain conditions are met	<code>while; for</code>



Thank you!
Questions?