

Counting in a Loop

```
zork = 0
print('Before', zork)
for thing in [9, 41, 12, 3, 74, 15] :
    zork = zork + 1
    print(zork, thing)
print('After', zork)
```

```
$ python countloop.py
```

```
Before 0
```

```
1 9
```

```
2 41
```

```
3 12
```

```
4 3
```

```
5 74
```

```
6 15
```

```
After 6
```

To **count** how many times we execute a loop, we introduce a **counter variable** that starts at 0 and we add **one** to it each time through the loop.

Summing in a Loop

```
zork = 0
print('Before', zork)
for thing in [9, 41, 12, 3, 74, 15] :
    zork = zork + thing
    print(zork, thing)
print('After', zork)
```

```
$ python countloop.py
```

```
Before 0
```

```
9 9
```

```
50 41
```

```
62 12
```

```
65 3
```

```
139 74
```

```
154 15
```

```
After 154
```

To **add up** a **value** we encounter in a loop, we introduce a **sum variable that starts at 0** and we add the **value** to the sum each time through the loop.

Finding the Average in a Loop

```
count = 0
sum = 0
print('Before', count, sum)
for value in [9, 41, 12, 3, 74, 15] :
    count = count + 1
    sum = sum + value
    print(count, sum, value)
print('After', count, sum, sum / count)
```

```
$ python averageloop.py
```

```
Before 0 0
```

```
1 9 9
```

```
2 50 41
```

```
3 62 12
```

```
4 65 3
```

```
5 139 74
```

```
6 154 15
```

```
After 6 154 25
```

An **average** just combines the **counting** and **sum** patterns and divides when the loop is done.

Filtering in a Loop

```
print('Before')
for value in [9, 41, 12, 3, 74, 15] :
    if value > 20:
        print 'Large number',value
print('After')
```

```
$ python search1.py
```

Before

Large number 41

Large number 74

After

We use an **if** statement in the **loop** to catch / filter the values we are looking for.

Search Using a Boolean Variable

```
found = False
print('Before', found)
for value in [9, 41, 12, 3, 74, 15] :
    if value == 3 :
        found = True
    print(found, value)
print('After', found)
```

```
$ python search1.py
```

```
Before False
```

```
False 9
```

```
False 41
```

```
False 12
```

```
True 3
```

```
True 74
```

```
True 15
```

```
After True
```

If we just want to search and **know if a value was found**, we use a **variable** that starts at **False** and is set to **True** as soon as we **find** what we are looking for.

How to Find the Smallest Value

```
largest_so_far = -1
print('Before', largest_so_far)
for the_num in [9, 41, 12, 3, 74, 15] :
    if the_num > largest_so_far :
        largest_so_far = the_num
    print(largest_so_far, the_num)

print('After', largest_so_far)
```

\$ python largest.py

Before -1

9 9

41 41

41 12

41 3

74 74

74 15

After 74

How would we change this to make it find the smallest value in the list?

Finding the Smallest Value

```
smallest_so_far = -1
print('Before', smallest_so_far)
for the_num in [9, 41, 12, 3, 74, 15] :
    if the_num < smallest_so_far :
        smallest_so_far = the_num
    print(smallest_so_far, the_num)

print('After', smallest_so_far)
```

We switched the variable name to `smallest_so_far` and switch the `>` to `<`

Finding the Smallest Value

```
smallest_so_far = -1
print('Before', smallest_so_far)
for the_num in [9, 41, 12, 3, 74, 15] :
    if the_num < smallest_so_far :
        smallest_so_far = the_num
    print(smallest_so_far, the_num)

print('After', smallest_so_far)
```

```
$ python smallbad.py
```

```
Before -1
```

```
-1 9
```

```
-1 41
```

```
-1 12
```

```
-1 3
```

```
-1 74
```

```
-1 15
```

```
After -1
```

We switched the variable name to `smallest_so_far` and switch the `>` to `<`

Finding the Smallest Value

```
smallest = None
print('Before')
for value in [9, 41, 12, 3, 74, 15] :
    if smallest is None :
        smallest = value
    elif value < smallest :
        smallest = value
    print(smallest, value)
print('After', smallest)
```

```
$ python smallest.py
```

```
Before
```

```
9 9
```

```
9 41
```

```
9 12
```

```
3 3
```

```
3 74
```

```
3 15
```

```
After 3
```

We still have a variable that is the **smallest** so far. The first time through the loop **smallest** is **None**, so we take the first **value** to be the **smallest**.

The “is” and “is not” Operators

```
smallest = None
print('Before')
for value in [3, 41, 12, 9, 74, 15] :
    if smallest is None :
        smallest = value
    elif value < smallest :
        smallest = value
    print smallest, value

print('After', smallest)
```

- Python has an **is** operator that can be used in logical expressions
- Implies “is the same as”
- Similar to, but stronger than **==**
- **is not** also is a logical operator

Summary

- While loops (indefinite)
- Infinite loops
- Using break
- Using continue
- For loops (definite)
- Iteration variables
- Loop idioms
- Largest or smallest



Acknowledgements / Contributions



These slides are Copyright 2010- Charles R. Severance
(www.dr-chuck.com) of the University of Michigan School of
Information and open.umich.edu and made available under a
Creative Commons Attribution 4.0 License. Please maintain this
last slide in all copies of the document to comply with the
attribution requirements of the license. If you make a change,
feel free to add your name and organization to the list of
contributors on this page as you republish the materials.

Initial Development: Charles Severance, University of Michigan
School of Information

... Insert new Contributors and Translators here

...