

LGT3109

Introduction to Coding for Business with Python (week 5)

Xiaoyu Wang
Dept. of LMS, The HK PolyU

Summary of Week 4

- Basic Concepts:

- Header `def name(arg):`
- Body `indented statement`
- Call function

- Built-in Functions

- Examples: `max`, `min`
- Package/Module: `math.pi/sqrt`
- `dir/help`

- User-defined functions

- Function like a detour
- `Define before using`
- Positional/keyword arg

Iterations

- Definite loop
- Indefinite loop
- Break/continue
- More examples

Motivation Case

- Foxconn wants to know the total shipping cost for the following six months.
- Shipping rate:
 - 1000HKD/container
 - 10% off for $100 \leq \text{containers} < 200$
 - 20% off for $\text{containers} \geq 200$

Input:

```
Enter shipping quantity for month 1: 150
Enter shipping quantity for month 2: 160
Enter shipping quantity for month 3: 240
Enter shipping quantity for month 4: 250
Enter shipping quantity for month 5: 180
Enter shipping quantity for month 6: 170
```

Output:

```
Total cost: 986000.0
```

Motivation Case-Code

- We can repeat the code 6 times.
- Drawbacks: (even with functions!)
- Code is too long
- Hard to debug

```
def compute_cost(quantity):  
    cost = 1000.0 * quantity  
    if quantity < 200 and quantity >= 100:  
        cost = cost * (1.0 - 0.1)  
    if quantity >= 200:  
        cost = cost * (1.0 - 0.2)  
    return cost
```

```
total_cost = 0.0
```

```
quantity = int(input('Enter shipping quantity for month 1: '))  
cost = compute_cost(quantity)  
total_cost = total_cost + cost
```

```
quantity = int(input('Enter shipping quantity for month 2: '))  
cost = compute_cost(quantity)  
total_cost = total_cost + cost
```

```
quantity = int(input('Enter shipping quantity for month 3: '))  
cost = compute_cost(quantity)  
total_cost = total_cost + cost
```

```
quantity = int(input('Enter shipping quantity for month 4: '))  
cost = compute_cost(quantity)  
total_cost = total_cost + cost
```

```
quantity = int(input('Enter shipping quantity for month 5: '))  
cost = compute_cost(quantity)  
total_cost = total_cost + cost
```

```
quantity = int(input('Enter shipping quantity for month 6: '))  
cost = compute_cost(quantity)  
total_cost = total_cost + cost
```

```
print('Total cost:', total_cost)
```

Motivation Case-Code

- Idea to simplify the code:

For each month of 1, 2, 3, 4, 5, 6, repeat the followings:

- Input quantity
- Call `cost = compute_cost(quantity)`
- Update `total_cost` by `total_cost + cost`

- We want a reusable component!

```
def compute_cost(quantity):  
    cost = 1000.0 * quantity  
    if quantity < 200 and quantity >= 100:  
        cost = cost * (1.0 - 0.1)  
    if quantity >= 200:  
        cost = cost * (1.0 - 0.2)  
    return cost
```

```
total_cost = 0.0
```

```
quantity = int(input('Enter shipping quantity for month 1: '))  
cost = compute_cost(quantity)  
total_cost = total_cost + cost
```

```
quantity = int(input('Enter shipping quantity for month 2: '))  
cost = compute_cost(quantity)  
total_cost = total_cost + cost
```

```
quantity = int(input('Enter shipping quantity for month 3: '))  
cost = compute_cost(quantity)  
total_cost = total_cost + cost
```

```
quantity = int(input('Enter shipping quantity for month 4: '))  
cost = compute_cost(quantity)  
total_cost = total_cost + cost
```

```
quantity = int(input('Enter shipping quantity for month 5: '))  
cost = compute_cost(quantity)  
total_cost = total_cost + cost
```

```
quantity = int(input('Enter shipping quantity for month 6: '))  
cost = compute_cost(quantity)  
total_cost = total_cost + cost
```

```
print('Total cost:', total_cost)
```

Loop

- Computers are often used to automate repetitive tasks.
- Python provides several loops:
 - for loop (definite loop)
 - while loop (indefinite loop)

Iterations

- Definite loop
- Indefinite loop
- Break/continue
- More examples

Loop-Definite Loop

- We want to loop through a **sequence (list)** of things (**items**)
 - A list of numbers, **a list of months**, etc.
- For **each item** in the **sequence**, we repeat the **process**.
- Sum up each number, **sum up the cost**, etc.
- These loops are called “definite loops ” because they execute an exact number of times: **each item in the list is iterated**.

Loop-Definite Loop-Simple Example

```
for i in [5, 4, 3, 2, 1] :  
    print(i)  
print('Blastoff!')
```

[5,4,3,2,1] indicates a list of numbers, 5, 4, 3, 2, 1
We will discuss “list” in more details in the future lectures

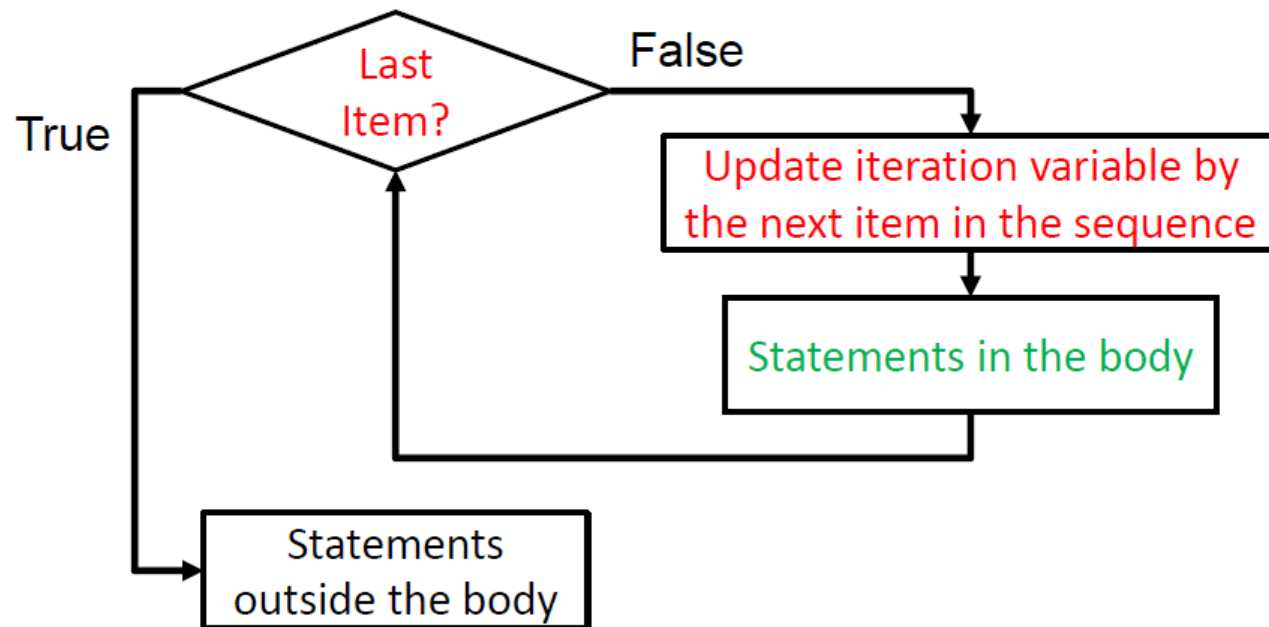
5
4
3
2
1
Blastoff!

Loop-Definite Loop-Structure

```
for iteration_variable in a sequence :  
    statements in the body
```

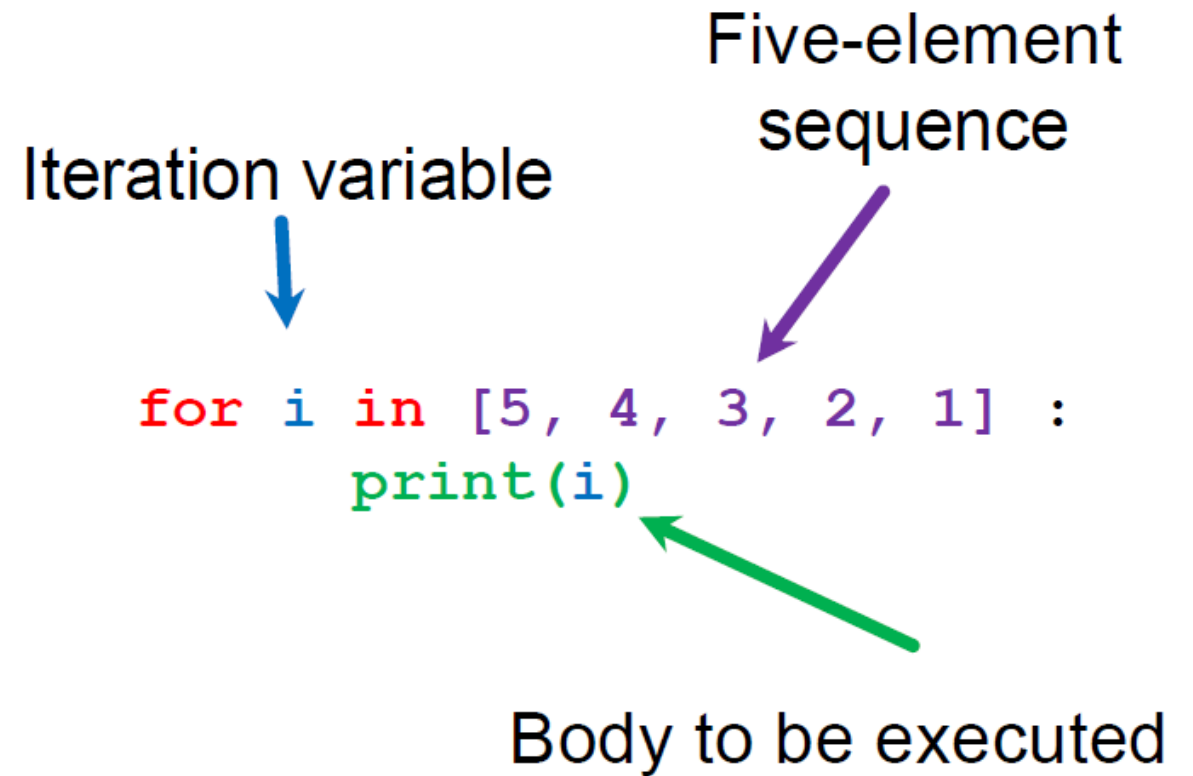
For each item in a sequence, repeat:

1. Store the item in the iteration variable, and
2. Execute the statements in the indented body



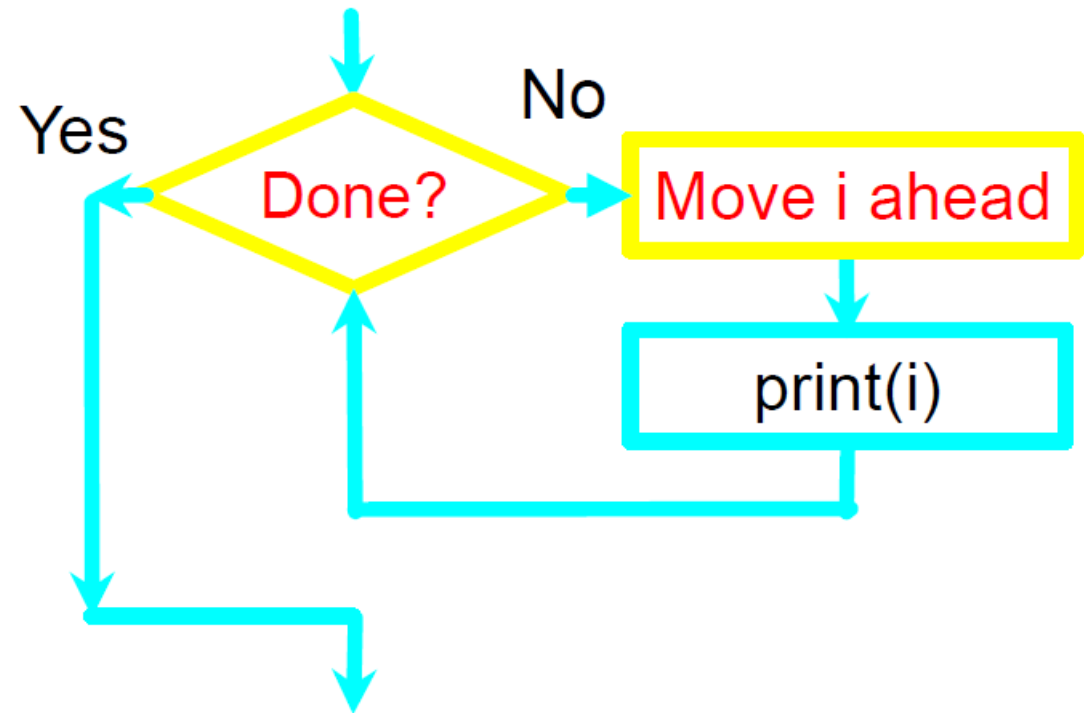
Loop-Definite Loop-Simple Example Recap

- The **iteration variable** iterates through the **sequence (ordered set)**.
- The **block (body)** of code is executed once for each item in the **sequence**.
- The **iteration variable** moves through all the items in the **sequence**.



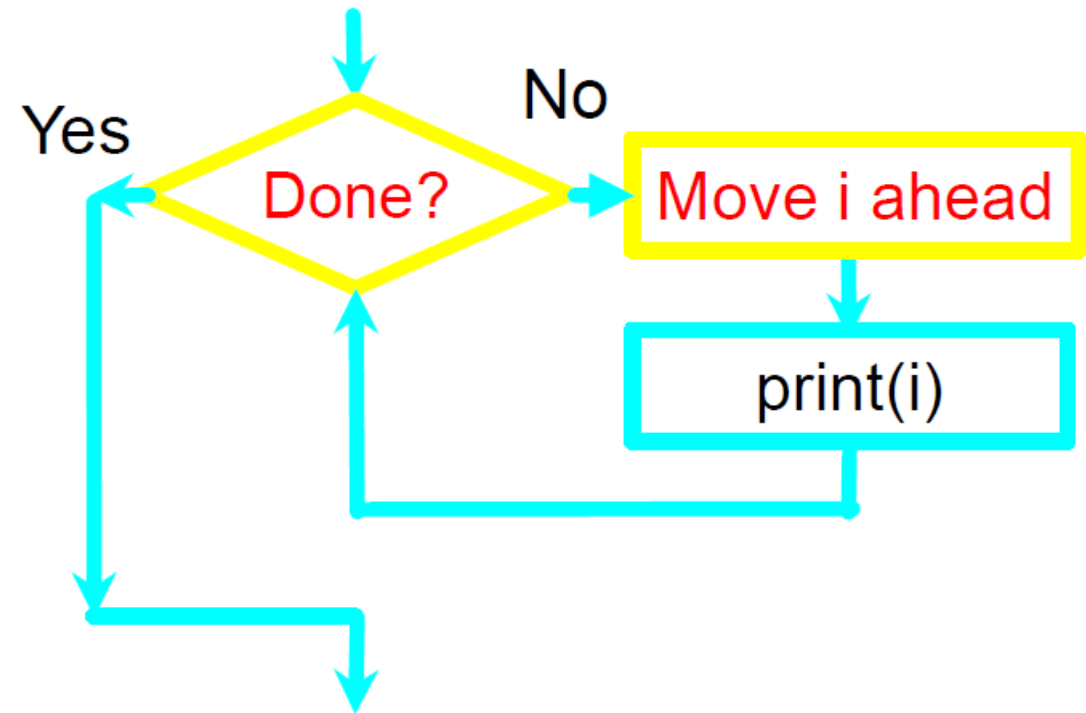
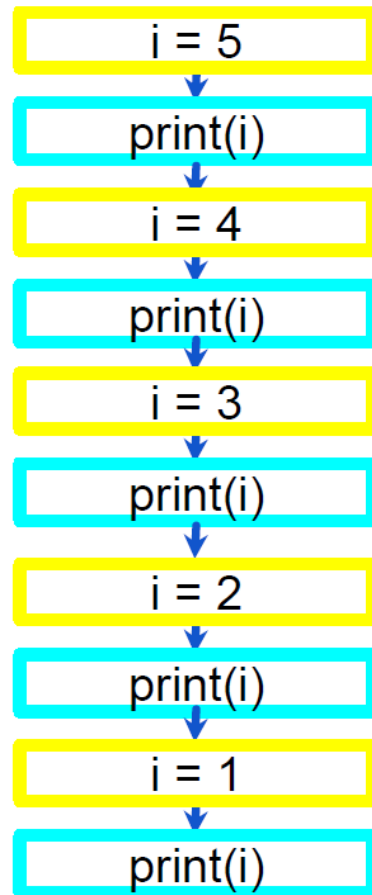
Loop-Definite Loop-Simple Example Recap

- The **iteration variable** iterates through the **sequence** (ordered set).
- The **block** (body) of code is executed once for each item in the **sequence**.
- The **iteration variable** moves through all the items in the **sequence**.



```
for i in [5, 4, 3, 2, 1] :  
    print(i)
```

Loop-Definite Loop-Simple Example Recap

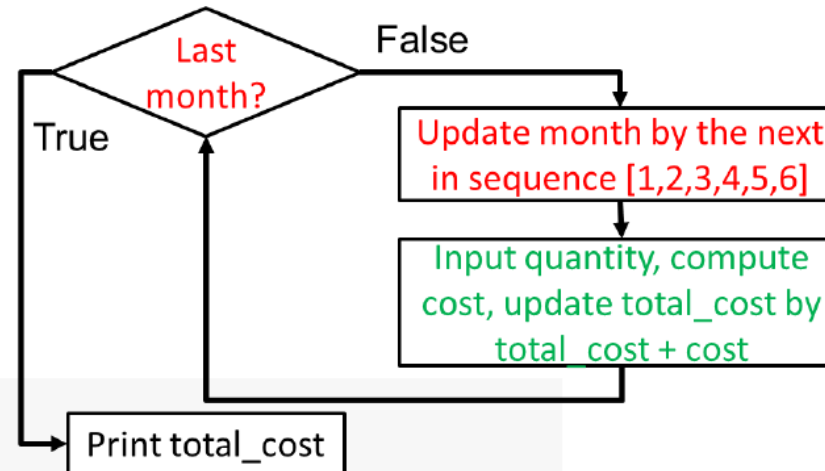


```
for i in [5, 4, 3, 2, 1] :  
    print(i)
```

Loop-Definite Loop-Motivation Case

- Use definite loop to iterate.

```
def compute_cost(quantity):  
    cost = 1000.0 * quantity  
    if quantity < 200 and quantity >= 100:  
        cost = cost * (1.0 - 0.1)  
    if quantity >= 200:  
        cost = cost * (1.0 - 0.2)  
    return cost  
  
total_cost = 0.0  
for month in [1,2,3,4,5,6]:  
    quantity = int(input('Enter shipping quantity for month '+str(month)+' : '))  
    cost = compute_cost(quantity)  
    total_cost = total_cost + cost  
  
print('Total cost:', total_cost)
```



Iterations

- Definite loop
- Indefinite loop
- Break/continue
- More examples

Loop-Indefinite Loop

- As long as the condition is satisfied, we repeat the **process**.
- Sum up each number, **sum up the cost**, etc.
- These loops are called “indefinite loops ” because they do not execute an exact number of times: **no list**.

Loop-Indefinite Loop-Simple Example

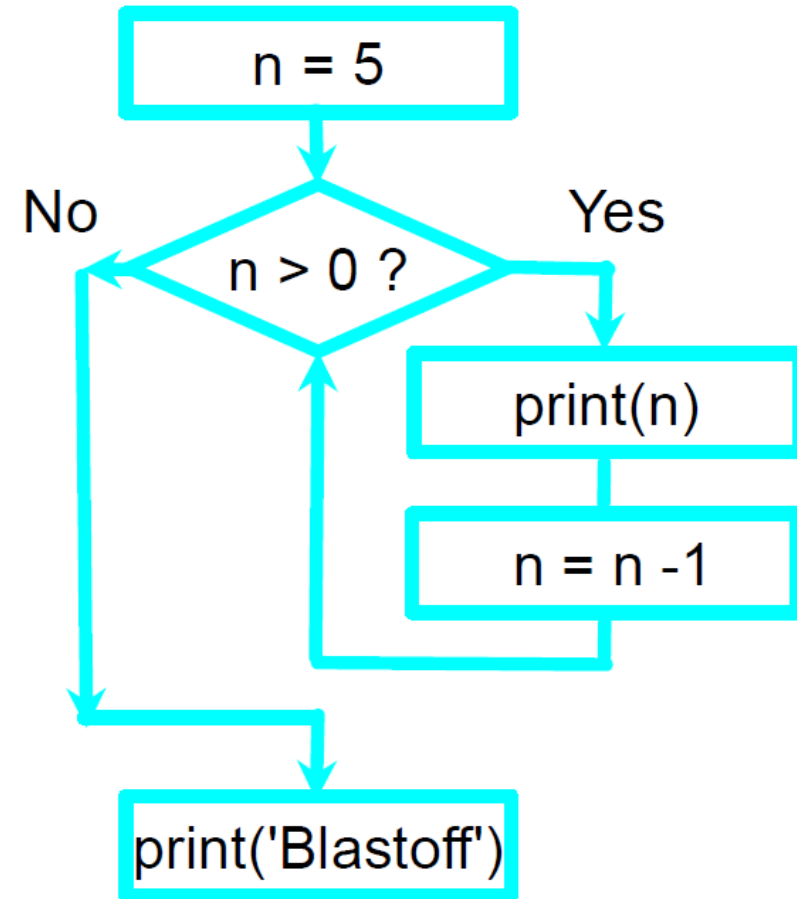
- When a logic **condition** is not satisfied, the loop stops.
- Loops (repeated steps) have **iteration variables** that are updated each time through a loop.

Program:

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

Output:

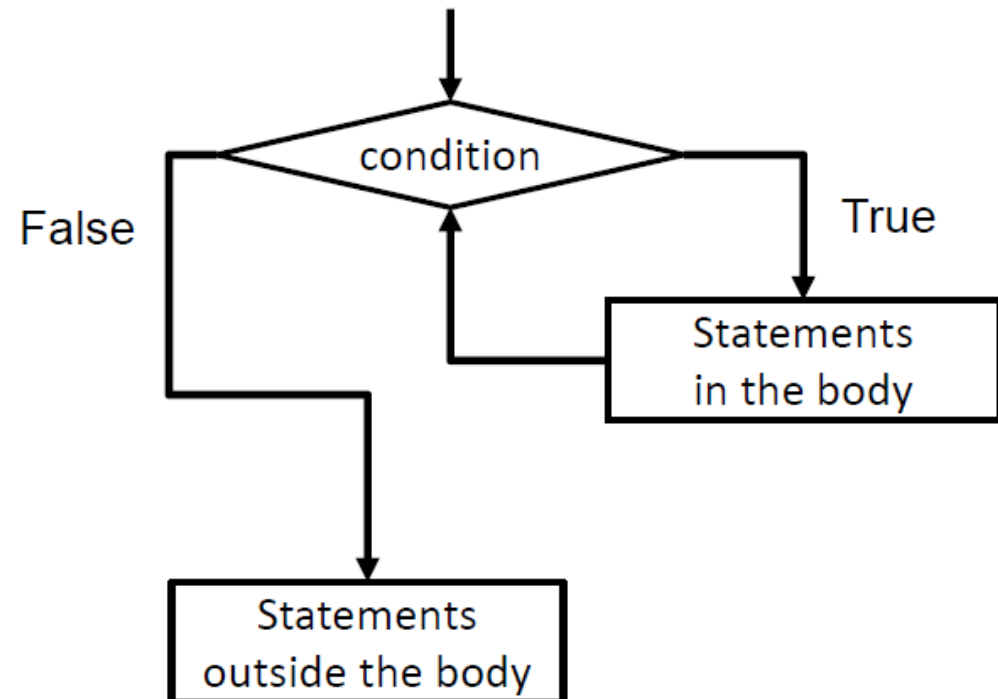
5
4
3
2
1
Blastoff!
0



Loop-Indefinite Loop-Structure

- Evaluate the condition, yielding **True** or **False**.
- If the **condition** is **False**, exit the while statement and continue to the next statement.
- If the **condition** is **True**, execute the statements in the (indented) body, and then go back to the **evaluation step**.

```
while condition :  
    statements in the body
```



Loop-Indefinite Loop-Updating

- The **body** of the loop needs to change the value of some **iteration variables**:
 - Updated in each **iteration** of the loop.
 - Controls when the loop finishes.
 - Eventually the **condition** becomes False, and the loop terminates.
- Before updating **an iteration variable**, it needs to be **initialized**.

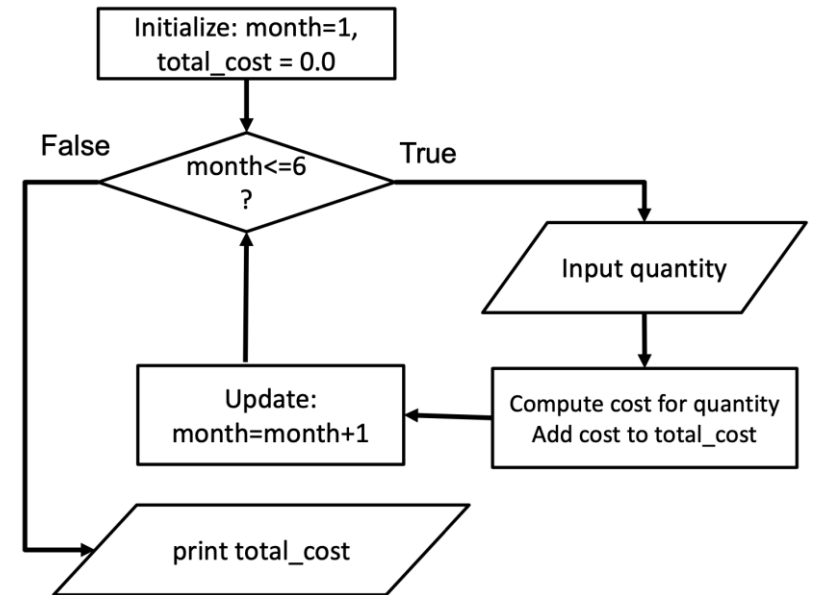
Program:

```
n = 5 #initialize
while n > 0 : #condition
    print(n)
    n = n - 1 #update
print('Blastoff!')
print(n)
```

Loop-Indefinite Loop-Motivation Case

- Use indefinite loop to iterate.

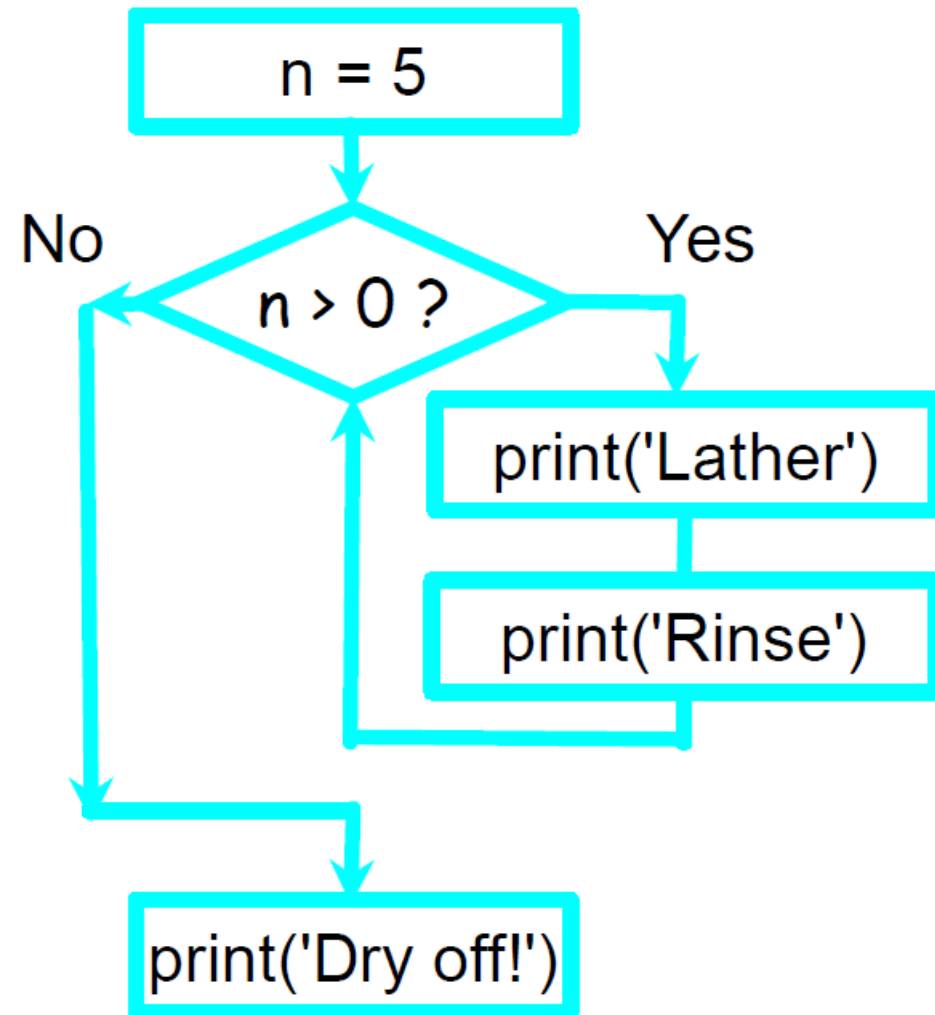
```
1 def compute_cost(quantity):
2     cost = 1000.0 * quantity
3     if quantity < 200 and quantity >= 100:
4         cost = cost * (1.0 - 0.1)
5     if quantity >= 200:
6         cost = cost * (1.0 - 0.2)
7     return cost
8
9 month = 1
10 total_cost = 0.0
11 while month <= 6:
12     quantity = int(input('Enter shipping quantity for month ' + str(month) + ': '))
13     cost = compute_cost(quantity)
14     total_cost = total_cost + cost
15     month = month + 1
16
17 print('Total cost:', total_cost)
```



Loop-Indefinite Loop-Problem

- What's wrong with this code?

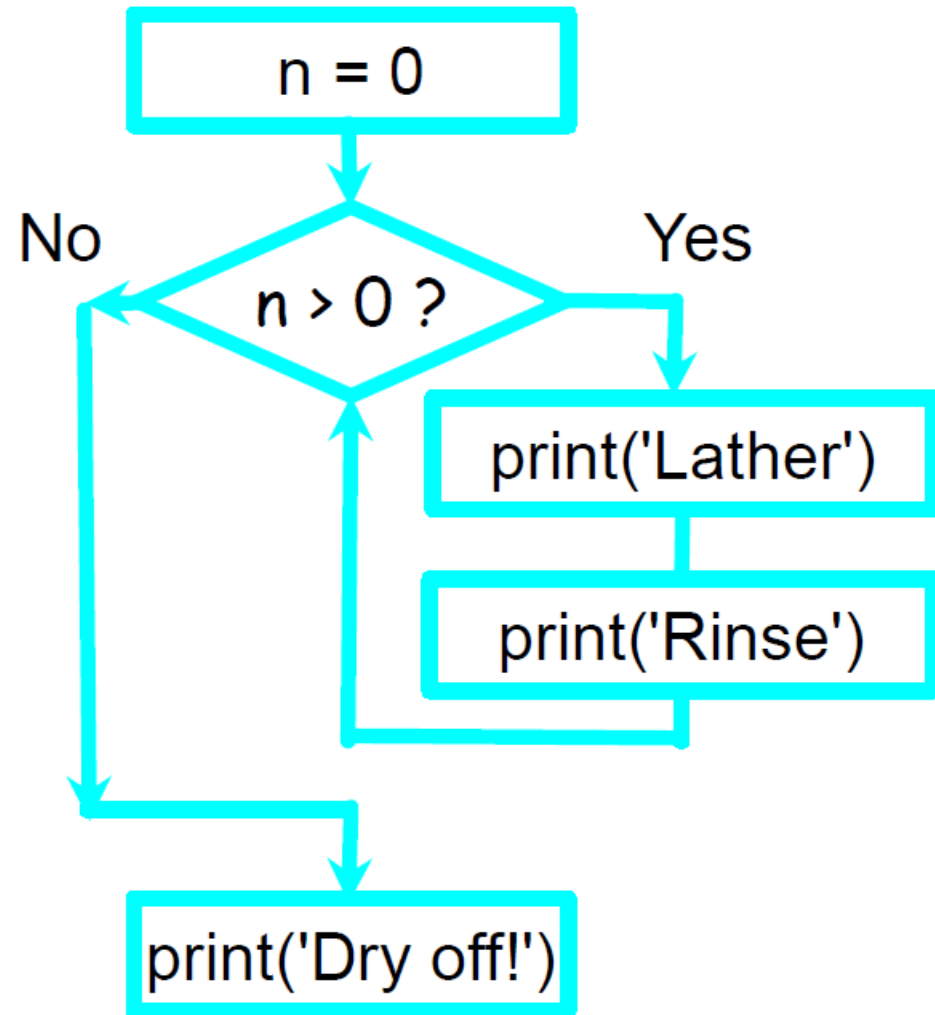
```
n = 5
while n > 0 :
    print('Lather')
    print('Rinse')
print('Dry off!')
```



Loop-Indefinite Loop-Problem

- What's wrong with this code?

```
n = 0
while n > 0 :
    print('Lather')
    print('Rinse')
    n = n + 1
print('Dry off!')
```



Loop-Indefinite Loop-Interruption

- In IDLE Python Shell:

➤ Press **Ctrl + C**.

```
>>> while True:
        print('*',end='')

*****
*****
*****Traceback (most recent call last):
  File "<pyshell#2>", line 2, in <module>
    print('*',end='')
KeyboardInterrupt
>>>
```

- In Jupyter Notebook:

➤ Click **Stop Button** in the Toolbar



Recap of Steps

- Python code step:

- *Sequential steps*

- *Repeated steps*

- *Conditional steps*

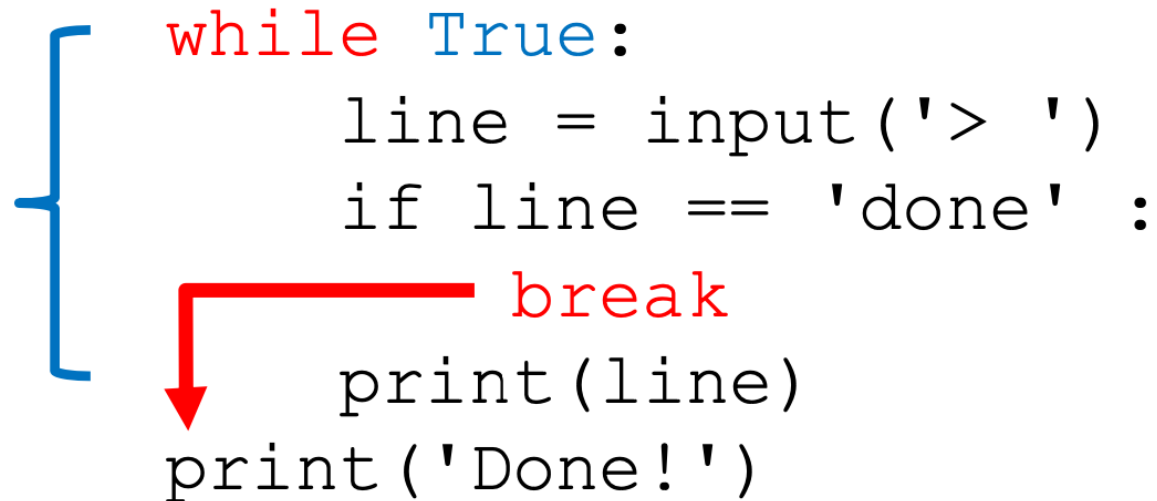
Iterations

- Definite loop
- Indefinite loop
- Break/continue
- More examples

Loop-Break

- The **break** statement ends the current loop and **jumps to the next statement**.
- It is like a loop test that can happen anywhere in the body of the loop.

Without **break**,
this is an infinite
loop




```
while True:
    line = input('> ')
    if line == 'done' :
        break
    print(line)
print('Done!')
```

```
> hello there
hello there
> finished
finished
> done
Done!
```

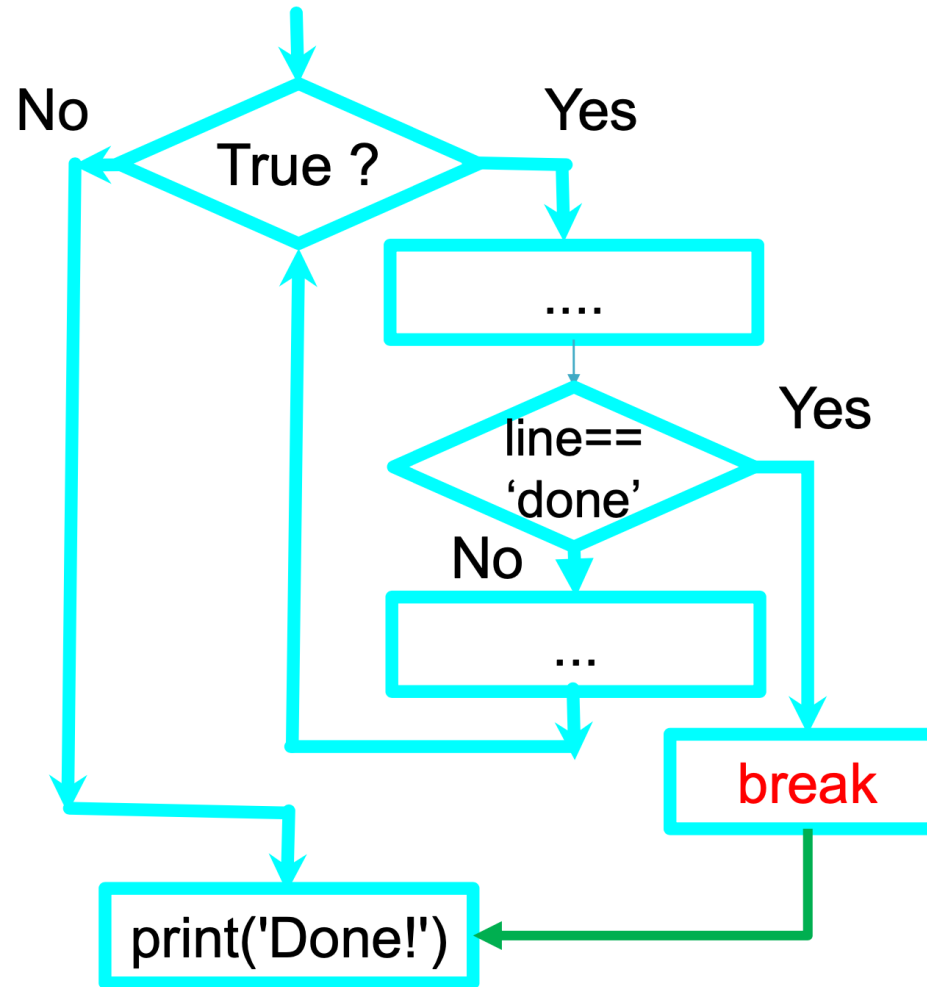
Loop-Break

```
while True:
    line = input('> ')
    if line == 'done' :
```

```
        break
    print(line)
print('Done!')
```



```
> hello there
hello there
> finished
finished
> done
Done!
```



Loop-Flag Variable

- Instead of using “break”, we can also use a variable as a flag to stop:
 - Indicating whether the stopping condition is satisfied or not.

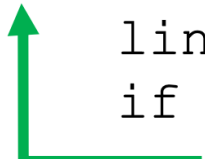
```
while True:
    line = input('> ')
    print(line)
    if line == 'done' :
        break
print('Done!')
```

```
flag_done = False
while not flag_done:
    line = input('> ')
    print(line)
    if line == 'done' :
        flag_done = True
print('Done!')
```

Loop-Continue

- The **continue** statement ends the current iteration and **jumps to the top of the loop** and **starts the next iteration**.

```
while True:
    line = input('> ')
    if line[0] == '#' :
        continue
    if line == 'done' :
        break
    print(line)
print('Done!')
```

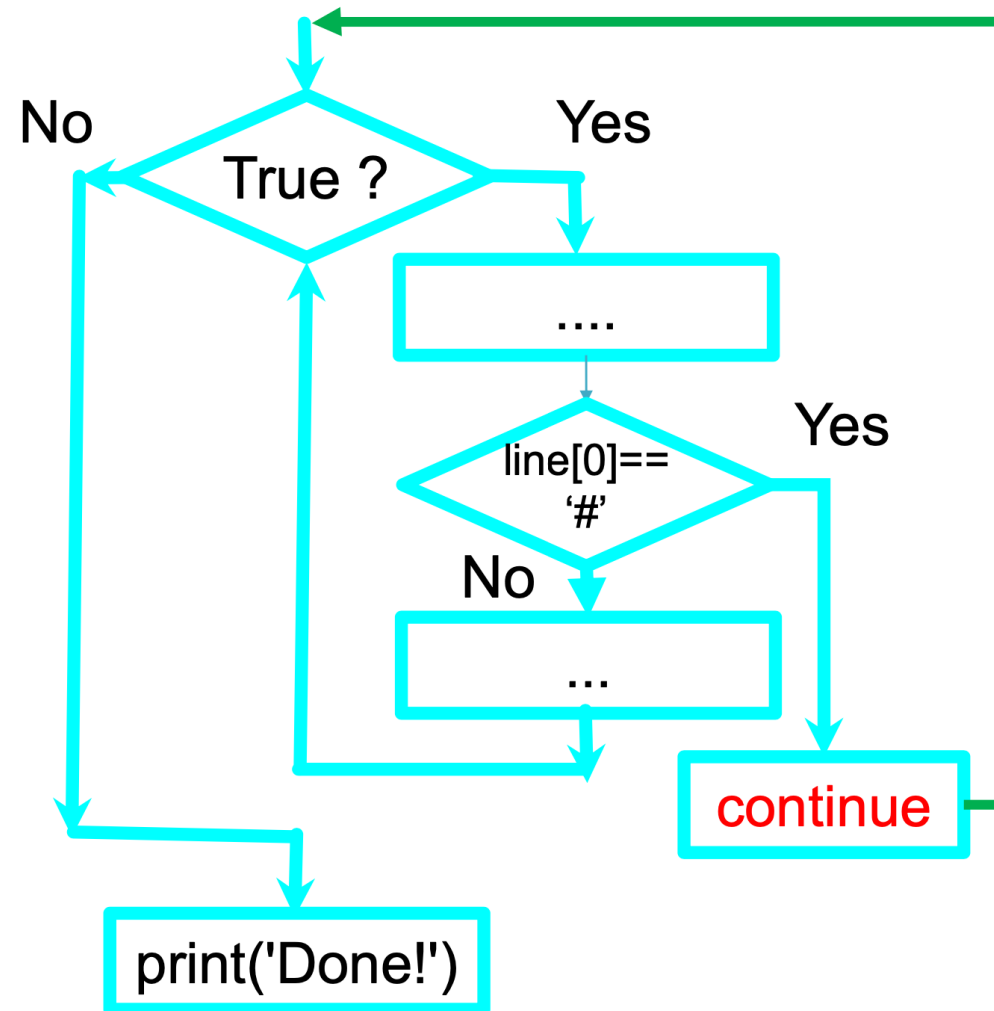


```
> hello there
hello there
> # don't print this
> print this!
print this!
> done
Done!
```

line[0] is the first character of the string line

Loop-Continue

```
while True:
    line = input('> ')
    if line[0] == '#' :
        continue
    if line == 'done' :
        break
    print(line)
print('Done!')
```



Loop-Using Break for Motivation Case

- Foxconn wants to know the **total shipping cost** for the following several months, as long as the input shipping quantity is **non-negative**.
- **Questions?**

```
total_cost = 0.0
month = 1
while True:
    quantity = int(input('Enter shipping quantity for month '+str(month)+' : '))
    if quantity < 0:
        break
    cost = compute_cost(quantity)
    total_cost = total_cost + cost
    month = month + 1

print('Total cost:', total_cost)
```

Loop-Comment

- For repeated loop, do not forget about the termination condition!
- Can use any repeated loop for the goal.

```
total_cost = 0
for month in [1,2,3,4,5,6]:
    quantity = int(input('Enter shipping quantity: '))
    cost = compute_cost(quantity)
    total_cost = total_cost + cost

print('Total cost:', total_cost)
```

```
total_cost = 0.0
month = 1
while month <= 6:
    quantity = int(input('Enter shipping quantity: '))
    cost = compute_cost(quantity)
    total_cost = total_cost + cost
    month = month + 1

print('Total cost:', total_cost)
```

```
total_cost = 0.0
month = 1
while True:
    quantity = int(input('Enter shipping quantity: '))
    if quantity < 0:
        break
    cost = compute_cost(quantity)
    total_cost = total_cost + cost
    month = month + 1

print('Total cost:', total_cost)
```

Iterations

- Definite loop
- Indefinite loop
- Break/continue
- More examples

Simple Example 2

- Looping through a set.

```
print('Before')
for entry in [9, 41, 12, 3, 74, 15] :
    #for each entry:
        print(entry)
print('After')
```

Before

9

41

12

3

74

15

After

Simple Example 2-Largest Number

- What's the largest number?

3 41 12 9 74 15

Simple Example 2-Largest Number

largest_so_far

-1

Simple Example 2-Largest Number

3

largest_so_far

3

Simple Example 2-Largest Number

3 41

largest_so_far

41

Simple Example 2-Largest Number

3 41 12

largest_so_far

41

Simple Example 2-Largest Number

3 41 12 9

largest_so_far

41

Simple Example 2-Largest Number

3 41 12 9 74

largest_so_far

74

Simple Example 2-Largest Number

3 41 12 9 74 15

largest_so_far

74

Simple Example 2-Largest Number

We make a **variable** that contains the largest value we have seen so far, and **initialize** it to be **-1**

We examine **each value in the given list**:

If the current value we are looking at is larger, it must be the new largest value we have seen so far, and the **variable** needs to be **updated**.

```
#Initialize the variable
largest_so_far = -1
print('Before', largest_so_far)
for the_num in [9, 41, 12, 3, 74, 15] :
    #for each entry:
        if the_num > largest_so_far :
            #update the variable
            largest_so_far = the_num
        print(largest_so_far, the_num)
#Output the variable
print('After', largest_so_far)
```

Before -1
9 9
41 41
41 12
41 3
74 74
74 15
After 74

Simple Example 2-Counting in a Loop

We make a **counting variable** that contains the number of values we have seen so far, and **initialize** it to be **0**.

We examine **each value in the given list**, and **update** the **counting variable** by **adding one to it** each time through the loop

```
#initialize the variable:
zork = 0
print('Before', zork)
for entry in [9, 41, 12, 3, 74, 15] :
    #for each entry:
        zork = zork + 1 #update the variable
        print(zork, entry)
#output the variable:
print('After', zork)
```

Before 0

1 9

2 41

3 12

4 3

5 74

6 15

After 6

Simple Example 2-Summing in a Loop

We make a **sum variable** that contains the number of values we have seen so far, and **initialize** it to be **0**.

We examine **each value in the given list**, and **update** the **sum variable** by **adding the value to it** each time through the loop

```
#initialize the variable:
zork = 0
print('Before', zork)
for entry in [9, 41, 12, 3, 74, 15] :
    #for each entry:
        zork = zork + entry #update the variable
        print(zork, entry)
#output the variable:
print('After', zork)
```

```
$ python countloop.py
Before 0
9 9
50 41
62 12
65 3
139 74
154 15
After 154
```

Simple Example 2-Average in a Loop

#initialize the variables:

```
count = 0
```

```
sum = 0
```

```
print('Before', count, sum)
```

```
for value in [9, 41, 12, 3, 74, 15] :
```

#for each entry, update the variables:

```
    count = count + 1
```

```
    sum = sum + value
```

```
    print(count, sum, value)
```

#output the variables:

```
print('After', count, sum, sum / count)
```

Before 0 0

1 9 9

2 50 41

3 62 12

4 65 3

5 139 74

6 154 15

After 6 154 25.666

Simple Example 2-Filtering in a Loop

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

Before
Large number 41
Large number 74
After

Simple Example 2-Search: Boolean Variable

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

```
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)
```

```
Before False
False 9
False 41
False 12
True 3
True 74
True 15
After True
```

How to make the code more efficient (i.e., reducing the number of iterations)?

Simple Example 2-Search: Boolean Variable

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

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

Before False
False 9
False 41
False 12
After True

How to make the code more efficient (i.e., reducing the number of iterations)? Use “break”

Simple Example 2-Smallest Number

- We know how to find the largest number.
- How about finding the smallest number.
- Modify the code below.

```
#Initialize the variable
largest_so_far = -1
print('Before', largest_so_far)
for the_num in [9, 41, 12, 3, 74, 15] :
    #for each entry:
        if the_num > largest_so_far :
            #update the variable
            largest_so_far = the_num
        print(largest_so_far, the_num)
#Output the variable
print('After', largest_so_far)
```

Before -1
9 9
41 41
41 12
41 3
74 74
74 15
After 74

Simple Example 2-Smallest Number

- Can we switch the variable name to `smallest_so_far` and switched the `>` to `<`?

```
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)
```

Simple Example 2-Smallest Number

- Can we switch the variable name to `smallest_so_far` and switched the `>` to `<`?

```
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)
```

Before -1

-1 9

-1 41

-1 12

-1 3

-1 74

-1 15

After -1

*Not correct, as we
expect the result to
be 3*

Simple Example 2-Smallest Number

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.

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

Before None

9 9

9 41

9 12

3 3

3 74

3 15

After 3

Remark: The **None** keyword is used to define a null value, or no value at all. None is a data type of its own (NoneType) and only None can be None.

Simple Example 2-Smallest Number

- The reserved keyword **None** is used to define a null value, or no value at all.
- How about using 10000?
- Think about finding the largest number. **None**

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

Acknowledgement

- Acknowledgements / Contributions
- These slides are Copyright 2010-Charles R. Severance (www.dr-chuck.com) of the University of Michigan School of Information 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
- Further Development: Zhou Xu, Hong Kong Polytechnic University
- Continuous development: Xiaoyu Wang, Hong Kong Polytechnic University