Lecture 4 Loop Structure

IDS 400
Programming for Data Science in Business

Lab Sessions

Lab 3 is available on Blackboard.

Assignment

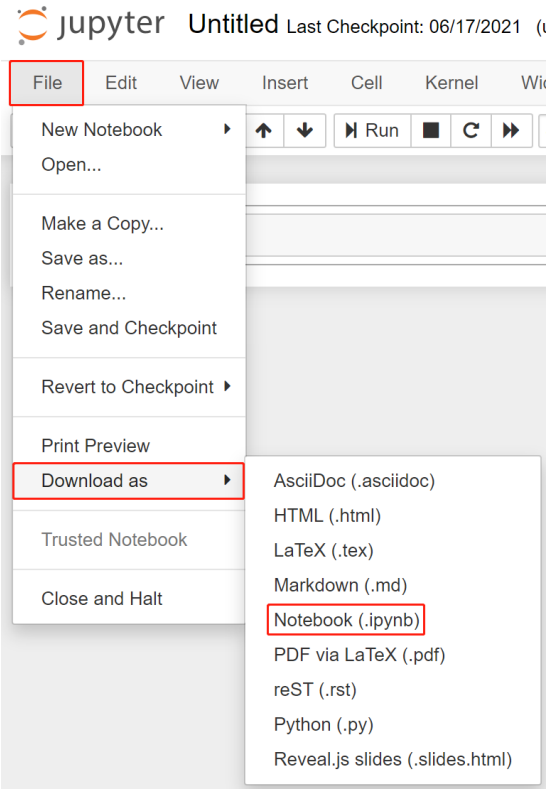
Assignment 2 is available on Blackboard.

- 1 week to work on it.

Submission

- This is no “fixed” solution for a task.
- Try to explore multiple solutions.

Please submit .ipynb and PDF file, not .py file.



Tentative schedule

Date	Lecture Number	Topics
08/24	Lecture 1	Introduction
08/31	Lecture 2	Basic
09/07	Lecture 3	Condition
09/14	Lecture 4	Loop
09/21	Lecture 5	String + **Quiz 1** → Online
09/28	Lecture 6	Type
10/05	Lecture 7	Function
10/12	Lecture 8	File + **Quiz 2** → Online
10/19	Lecture 9	Pandas
10/26	Lecture 10	Numpy
11/02	Lecture 11	Machine Learning
11/09	Lecture 12	Visualization
11/16	Lecture 13	Web Scraping & Deep Learning
11/23	*Thanksgiving*	*No lecture*
11/30	**Final presentation**	In class presentation
12/05	**Project submission due**	

Quiz 1

- **25 multichoice questions**
- **60 minutes**

Last class

- Conditional statement (*if* statement)

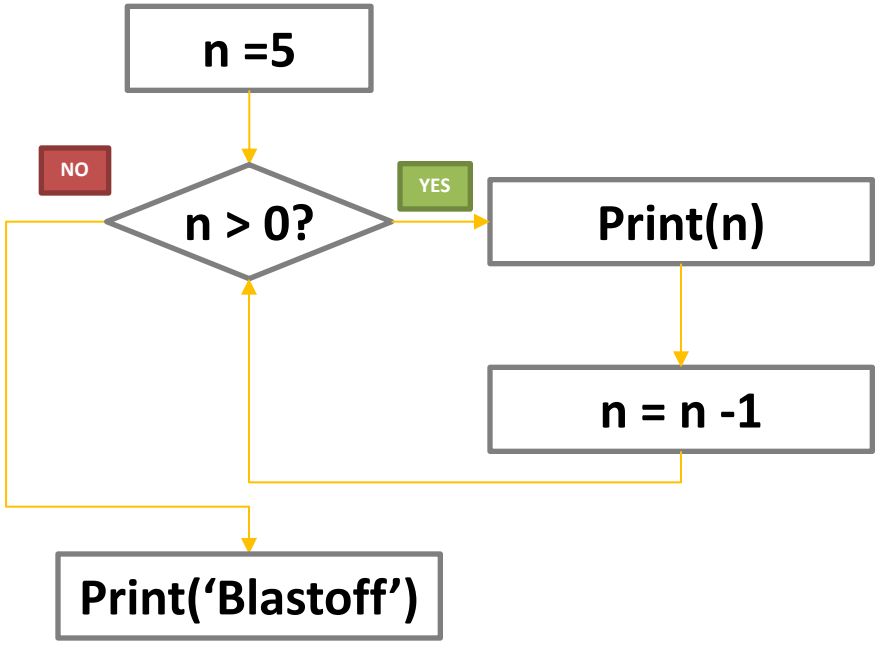
For This Class

- Loop (***while*** statement, ***for*** statement)

Repeated steps

- Computers are often used to automate repetitive tasks – **Loop**

```
5
4
3
2
1
Blastoff!
```



The while statement

- The syntax of ***while*** statement.

```
while <expr>:
    <statement>
    <statement>
    break
    <statement>
    <statement>
    continue
    <statement>
    <statement>

<statement>
```

The flow of execution:

a) Evaluate the expression, yielding True or False;

b) If the expression is True, execute each of the statements in the body and then go back to step (a)

c) If the expression is False, exit the entire while statement and continue execution at the next statement.

The while statement

- The syntax of ***while*** statement.

```
while <expr>:
    <statement>
    <statement>
    break
    <statement>
    <statement>
    continue
    <statement>
    <statement>
<statement>
```

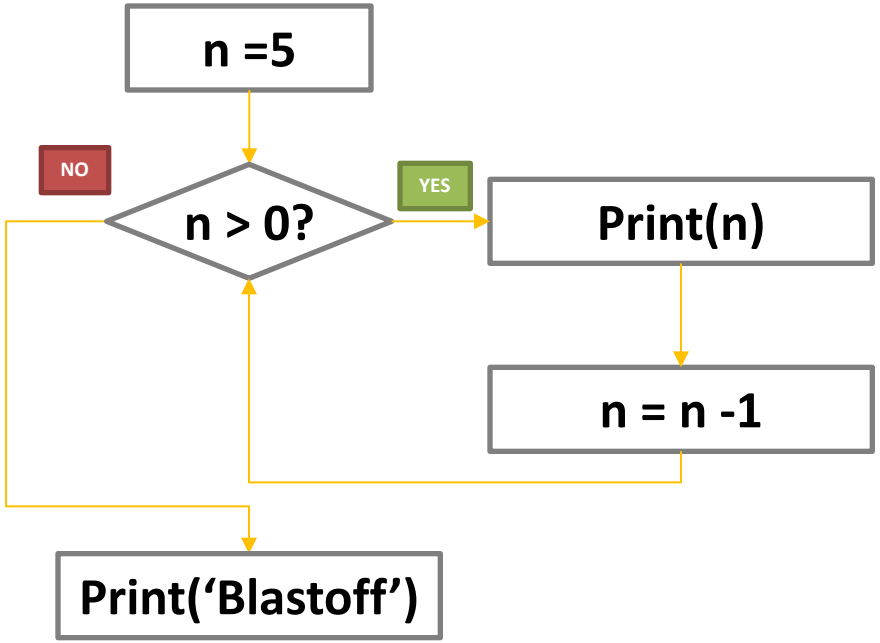
The flow of execution:

a) Evaluate the expression, yielding True or False;

b) If the expression is True, execute each of the statements in the body and then go back to step (a)

c) If the expression is False, exit the entire while statement and continue execution at the next statement.

Example

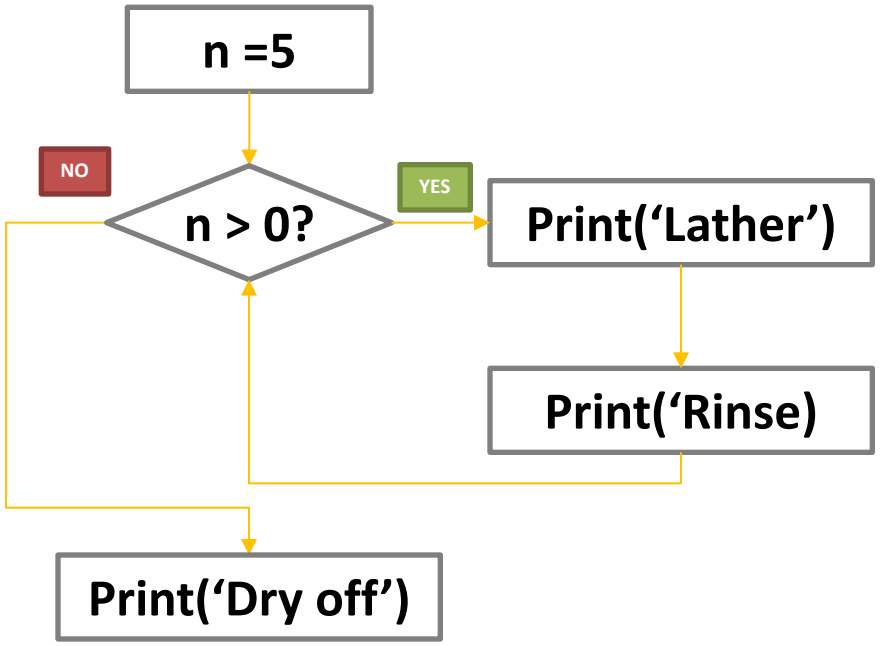



```python
n =5
while n > 0:
    print(n)
    n = n - 1
print('Blastoff!')
```

```
5
4
3
2
1
Blastoff!
```

An infinite loop

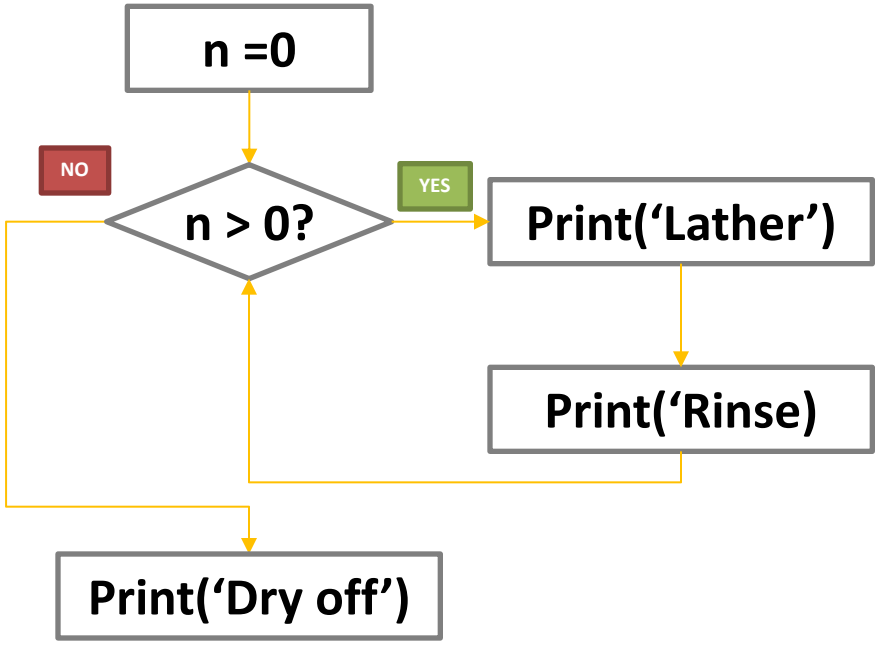
- What's wrong with this loop?




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

Another loop

- What does this loop do?




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

Making "smart" loops

```
# Set some variables to initial values

while expression:      # related to variables
    # 1. Look for something or do something
    #    in each iteration.
    # 2. Update variables
```

The nested while statement

- How should we start?
- What is the output?

```
x = 5
while x ! = 1:
    print(x)
    while x > 3:
        print('x>3')
        x = x - 1
    x = x - 1
```

The nested while statement

- How should we start?
- What is the output?

```
x = 5
while x ! = 1:
    print(x)
    while x > 3:
        print('x>3')
        x = x - 1
    x = x - 1
```

Innermost loop

The nested while statement

- What is the output?

```
x = 5
while x ! = 1:
    print(x)
    while x > 3:
        print('x>3')
        x = x - 1
    x = x - 1
```

Innermost loop

Output:
5
x>3
x>3
2

Breaking out of a loop

```
while <expr>:
    <statement>
    <statement>
     break
    <statement>
    <statement>
     continue
    <statement>
    <statement>

<statement>
```

- The ***break*** statement ends the current innermost loop and *jumps to the statement immediately following the loop*.
- It can happen anywhere in the body of the loop, depending on your needs.
- All statements in the loop body and after break will NOT be executed if break happens.

Example

```
x = 5
while x > 0:
    print(x)
    if x == 3:
        break
    x = x - 1
print(x)
```

Example

```
x = 5
while x > 0:
    print(x)
    if x == 3:
        break
    x = x - 1
print(x)
```

Block of While-loop

Block of If-statement

Example

```
x = 5
while x > 0:
    print(x)
    if x == 3:
        break
    x = x - 1
print(x)
```

Block of While-loop

Block of If-statement

Output:
5
4
3
3

Example

```
x = 5
while x > 0:
    print(x)
    if x == 3:
        break
    x = x - 1
print(x)
```

```
x = 5
while x > 2:
    print(x)
    while True:
        print('x>2')
        if x == 3:
            break
        x = x - 1
    x = x - 1
print(x)
```

Output:
5
4
3
3

Example

```
x = 5
while x > 0:
    print(x)
    if x == 3:
        break
    x = x - 1
print(x)
```

Output:
5
4
3
3

```
x = 5
while x > 2:
    print(x)
    while True:
        print('x>2')
        if x == 3:
            break
        x = x – 1
    x = x - 1
print(x)
```

Innermost loop

Output:
5
x>2
x>2
x>2
2

The continue statement

```
while <expr>:
    <statement>
    <statement>
    break
    <statement>
    <statement>
    continue
    <statement>
    <statement>

<statement>
```

- The ***continue*** statement *ends the current iteration of the innermost loop* and *jumps to the top of the loop and starts the next iteration*.
- It can happen anywhere in the body of the loop, depending on your needs.

Example

```
x = 5
while x > 0:
    x = x – 1
    if x == 3:
        continue
    print(x)
print(x)
```

Example

```
x = 5
while x > 0:
    x = x – 1
    if x == 3:
        continue
    print(x)
print(x)
```

Output:
4
2
1
0
0

Example

```
x = 5
while x > 0:
    x = x – 1
    if x == 3:
        continue
    print(x)
print(x)
```

```
x = 5
while x > 2:
    print(x)
    while x > 0:
        x = x – 1
        if x < 3:
            continue
            print('x<3')
        else:
            print('x>=3')
    x = x - 1
print(x)
```

Output:
4
2
1
0
0

Example

```
x = 5
while x > 0:
    x = x – 1
    if x == 3:
        continue
    print(x)
print(x)
```

Output:
4
2
1
0
0

```
x = 5
while x > 2:
    print(x)
    while x > 0:
        x = x – 1
        if x < 3:
            continue
            print('x<3')
        else:
            print('x>=3')
    x = x - 1
print(x)
```

Innermost loop

Output:
5
x>=3
x>=3
-1

Indefinite loop

- ***While*** loops are called "***indefinite loops***" because they keep going until a logical expression becomes False.
- The loops we have seen so far are easy to examine to see if they will terminate or if they are "infinite loops".
- Sometimes it is harder to be sure if a loop will terminate.

Definite loop

- Quite often we have a list of items – effectively **a finite set of things**.
- We can write a loop to run the loop once for each of the items in a set using the Python ***for*** construct.
- These loops are called "definite loops" because they execute *an exact number* of times.
- We say that "*definite loops iterate through the members of a set*".

The for statement

- The expression list is evaluated once; it should yield an iterable object (e.g., list, tuple, etc.)
- For each member in the expression_list, execute all statements in the for body.
- ***Break*** and ***continue*** statements are also applicable in *for* loops.
- The syntax of *for* statement:

```
for iterator in expression_list:
    <statement>
    <statement>
    …
<statement>
```

The for statement

- The **iteration variable** “iterates” though the **sequence** (ordered set).
- The block (body) of code is executed **once for each element in the sequence**.
- The iteration variable moves through **all of the values in the sequence**.

Iteration variable

Five-element sequence

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

The for statement

- Definite loops (*for* loops) have explicit iteration variables that change each time through a loop. These iteration variables move through the sequence or set.

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

Output:
5
4
3
2
1
Blastoff

Example

```
for i in [5, 4, 3, 2, 1]:
    if i % 2 == 0:
        print(i, ":even" )
    else:
        print(i, ":odd" )
print('Blastoff')
```

Output:

5:odd

4:even

3:odd

2:even

1 :odd

Blastoff

Nested for statement

- Consider the “inner for loop” as “one statement” within the outer loop body.
- For each member in the “outer loop”, execute all statements.
- When execute inner for loop statement, consider it as a real loop.

```
for iterator in expression_list:
    <statement>
    <statement>
     for iterator in expression_list:
        <statement>
        <statement>
     <statement> (outer for)
<statement> (after outer for)
```

Nested for statement

- Consider the “inner for loop” as “one statement” within the outer loop body
- For each member in the “outer loop”, execute all statements.
- When execute inner for loop statement, consider it as a real loop.

```
for iterator in expression_list:
    <statement>
    <statement>
     for iterator in expression_list:
         <statement>
         <statement>
     <statement> (outer for)
<statement> (after outer for)
```

Inner for statement

Nested for statement

- Consider the “inner for loop” as “one statement” within the outer loop body
- For each member in the “outer loop”, execute all statements.
- When execute inner for loop statement, consider it as a real loop.

```
for iterator in expression_list:
    <statement>
    <statement>
    for iterator in expression_list:
        <statement>
        <statement>
    <statement> (outer for)
<statement> (after outer for)
```

Outer for statement

Inner for statement

Example

```
for i in [1, 2, 3]:
    for j in [1, 2, 3]:
        print (i * j)
print ('Blastoff')
```

Output:
1
2
3
2
4
6
3
6
9
Blastoff

Example

```
for i in [1, 2, 3]:
    j = 1
    while j <= i:
        print (i)
        j = j + 1
print ('Blastoff')
```

Output:
1
2
2
3
3
3
Blastoff

Example

- Given a list of numbers, what is the largest number?

3 41 12 9 74 15

How to solve this problem?
Should we use for loop or while loop?

What is the largest number

```
# Set some variables to initial values
Largest_so_far = 0

for current in [3, 41, 12, 9, 74, 15]:
    # 1. Look for something or do something
    #    to each  element separately.
    if current > largest_so_far:
    # 2. Update a variable
        largest_so_far = current

# Look at the variables.
print (largest_so_far)
```

largest_so_far: 0 → 3 → 41 → 74

Counting in a loop

- 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.

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

Output:
Before 0
1 9
2 41
3 12
4 3
5 74
6 15
After 6

Summing in a loop

- 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 sum each time through the loop.

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

Output:
Before 0
9 9
50 41
62 12
65 3
139 74
154 15
After 154

Finding average in a loop

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

```
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)
```

```
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
```

Another example

	Col 1	Col 2	Col 3	Col 4	Col 5
Row 1	1	2	3	4	5
Row 2	2	4	6	8	10
Row 3	3	6	9	12	15
Row 4	4	8	12	16	20
Row 5	5	10	15	20	25

- Print the square where each element is the product of its row and column number (i*j).

Another example

	Col 1	Col 2	Col 3	Col 4	Col 5
Row 1	1	2	3	4	5
Row 2	2	4	6	8	10
Row 3	3	6	9	12	15
Row 4	4	8	12	16	20
Row 5	5	10	15	20	25

- Print the square where each element is the product of its row and column number (i*j).

```python
# i controls rows, and j controls columns

# start from the first row
i = 1

# define the maximum height of the square
height = 5

# for each row, we iterate all columns using nested loops
while i <= height:
    j = 1
    line = ''
    while j <= height:
        # For each row, concatenate columns and save it in a string
        line = line + str(i*j) + '\t'
        j += 1  # move to next column
    print(line)
    i += 1  # move to next row
```

```
1       2       3       4       5
2       4       6       8       10
3       6       9       12      15
4       8       12      16      20
5       10      15      20      25
```

Another example

	Col 1	Col 2	Col 3	Col 4	Col 5
Row 1	1	2	3	4	5
Row 2	2	4	6	8	10
Row 3	3	6	9	12	15
Row 4	4	8	12	16	20
Row 5	5	10	15	20	25

- Print the square where each element is the product of its row and column number (i*j).

```python
# i controls rows, and j controls columns

# start from the first row
i = 1

# define the maximum height of the square
height = 5

# for each row, we iterate all columns using nested loops
while i <= height:
    j = 1
    line = ''
    while j <= height:
        # For each row, concatenate columns and save it in a string
        line = line + str(i*j) + '\t'
        j += 1  # move to next column
    print(line)
    i += 1  # move to next row
```

```
1       2       3       4       5
2       4       6       8       10
3       6       9       12      15
4       8       12      16      20
5       10      15      20      25
```

	Col 1	Col 2	Col 3	Col 4	Col 5
Row 1	1				
Row 2	2	4			
Row 3	3	6	9		
Row 4	4	8	12	16	
Row 5	5	10	15	20	25

- Print the triangle where each element is the product of its row and column number (i*j).

Another example

	Col 1	Col 2	Col 3	Col 4	Col 5
Row 1	1	2	3	4	5
Row 2	2	4	6	8	10
Row 3	3	6	9	12	15
Row 4	4	8	12	16	20
Row 5	5	10	15	20	25

	Col 1	Col 2	Col 3	Col 4	Col 5
Row 1	1				
Row 2	2	4			
Row 3	3	6	9		
Row 4	4	8	12	16	
Row 5	5	10	15	20	25

- Print the square where each element is the product of its row and column number (i*j).

```python
# i controls rows, and j controls columns

# start from the first row
i = 1

# define the maximum height of the square
height = 5

# for each row, we iterate all columns using nested loops
while i <= height:
    j = 1
    line = ''
    while j <= height:
        # For each row, concatenate columns and save it in a string
        line = line + str(i*j) + '\t'
        j += 1  # move to next column
    print(line)
    i += 1  # move to next row
```

```
1       2       3       4       5
2       4       6       8       10
3       6       9       12      15
4       8       12      16      20
5       10      15      20      25
```

- Print the triangle where each element is the product of its row and column number (i*j).

```python
# i controls rows, and j controls columns

# start from the first row
i = 1

# define the maximum height of the square
height = 5

# for each row, we iterate all columns using nested loops
while i <= height:
    j = 1
    line = ''
    while j <= i:
        # For each row, concatenate columns and save it in a string
        line = line + str(i*j) + '\t'
        j += 1  # move to next column
    print(line)
    i += 1  # move to next row
```

```
1
2       4
3       6       9
4       8       12      16
5       10      15      20      25
```

Lab *Loop*