DATA SCIENCE 101: PYTHON ITERATIONS (PART 1)

AGENDA

- Recap of previous lessons
- Python iterations

RECAP OF PREVIOUS LESSON

- Decision structure & Boolean logic:
  - Sequence structure
  - Boolean operators (conditional & relational)
  - IF, ELIF, ELSE
  - Nested decisions
  - Try-Except
  - “Not” operator

- Collections:
  - What are collections?
  - Lists
  - Tuples
  - Dictionaries

INTRODUCTION

- Purpose of iterations?
- What are variable tasks?
- Types of iteration constructs (While & For)

PROBLEM: REPETITIVE TASKS

- Example of repetitive tasks: Just imagine if you were asked by your boss to write a program that can print “Hello World” one hundred times
- This is a simple program, but yet you are going to have to write 100 lines of codes just to render this ☹️

ANALYSIS OF PROBLEM

- By just using the current tools available at our disposal to deal with the previous problem, we face the following disadvantages:
    - We would have ended up with a long sequence of code just to solve the previous problems
    - Writing this kind of program can be time consuming
    - If duplicated part of code needs to be corrected, then the correction must be implemented many times

SOLUTION TO PROBLEM: ITERATION

- The solution to the above disadvantages would be:
  - Write the codes for the operation one time
  - Place the codes in a repetition structure for it to be repeated as many times as possible
- This repetition structure is actually call an iteration device or more commonly known as the “loop”
- There are different kinds of iteration device we can write in python: definite and indefinite loops

DEFINITE LOOPS

- Quite often we have a list of items or lines in a file - 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"

FOR LOOPS

- Applying for loops in fundamental questions (list as data type)
- Other ways to implement a loop (using range)
- Applying for loops in fundamental questions (dictionary as data type)

EXECUTION TRACE: A SIMPLE DEFINITE FOR LOOP*

```python
countdown = [5, 4, 3, 2, 1]

for i in countdown:
    print(i)
print('Peekaboo!')
```

EXECUTION TRACE: A SIMPLE DEFINITE FOR LOOP*

```
countdown = [5, 4, 3, 2, 1]

for i in countdown:
    print(i)
print('Peekaboo!')
```

i | 5

EXECUTION TRACE: A SIMPLE DEFINITE FOR LOOP*

```python
countdown = [5, 4, 3, 2, 1]

for i in countdown:
    print(i)
print('Peekaboo!')
```

i 5

Output:

```
5
```

EXECUTION TRACE: A SIMPLE DEFINITE FOR LOOP*

```python
countdown = [5, 4, 3, 2, 1]

for i in countdown:
    print(i)
print('Peekaboo!')
```

i 4

Output:

```
5
```

EXECUTION TRACE: A SIMPLE DEFINITE FOR LOOP*

```
countdown = [5, 4, 3, 2, 1]

for i in countdown:
    print(i)
print('Peekaboo!')
```

```
i  4
```

Output:

```
5
4
```

EXECUTION TRACE: A SIMPLE DEFINITE FOR LOOP*

```
i  3
```

```python
countdown = [5, 4, 3, 2, 1]

for i in countdown:
    print(i)
print('Peekaboo!')
```

```
Output:
5
4
```

EXECUTION TRACE: A SIMPLE DEFINITE FOR LOOP*

```
countdown = [5, 4, 3, 2, 1]

for i in countdown:
    print(i)
print('Peekaboo!')
```

i 3

Output:

```
5
4
3
```

EXECUTION TRACE: A SIMPLE DEFINITE FOR LOOP*

i 2

```
countdown = [5, 4, 3, 2, 1]

for i in countdown:
    print(i)
print('Peekaboo!')
```

Output:

```
5
4
3
```

EXECUTION TRACE: A SIMPLE DEFINITE FOR LOOP*

```
countdown = [5, 4, 3, 2, 1]

for i in countdown:
    print(i)
print('Peekaboo!')
```

i 2

Output:

```
5
4
3
2
```

EXECUTION TRACE: A SIMPLE DEFINITE FOR LOOP*

```
countdown = [5, 4, 3, 2, 1]

for i in countdown:
    print(i)
print('Peekaboo!')
```

i 1

Output:

```
5
4
3
2
```

EXECUTION TRACE: A SIMPLE DEFINITE FOR LOOP*

```
countdown = [5, 4, 3, 2, 1]

for i in countdown:
    print(i)
print('Peekaboo!')
```

i 1

```
Output:
5
4
3
2
1
```

EXECUTION TRACE: A SIMPLE DEFINITE FOR LOOP*

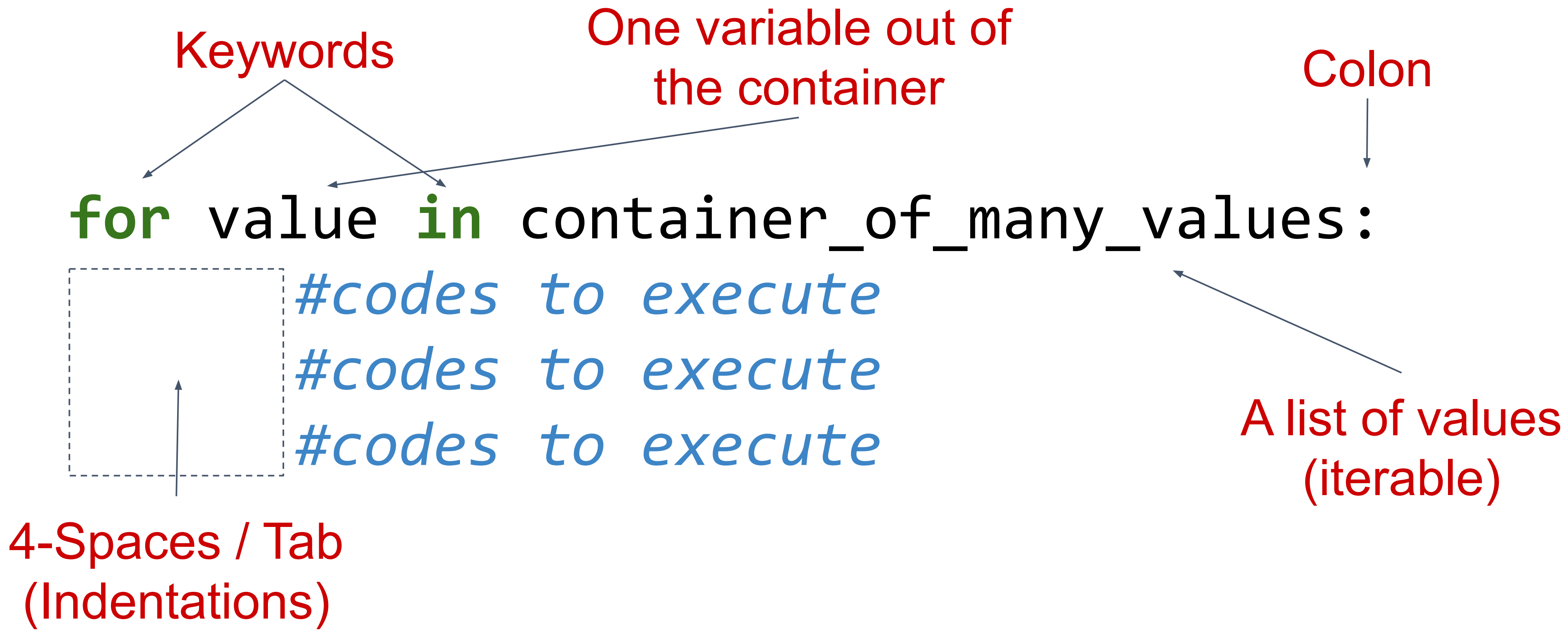
```
countdown = [5, 4, 3, 2, 1]

for i in countdown:
    print(i)
print('Peekaboo!')
```

i 1

Output:

```
5
4
3
2
1
Peekaboo!
```

SYNTAX - FOR LOOPS

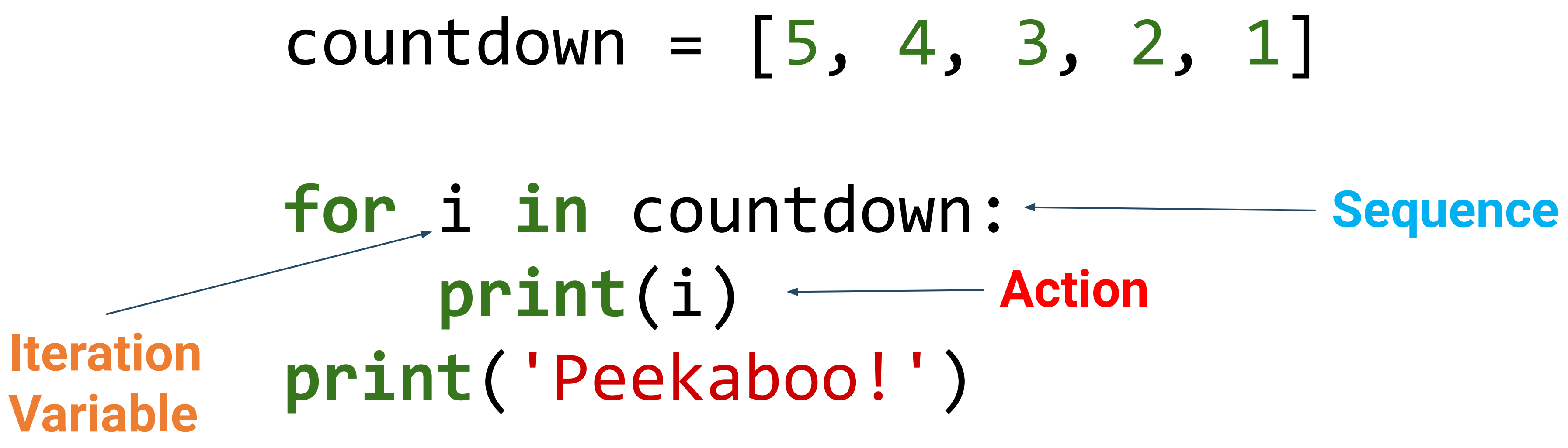


```
for value in container_of_many_values:
    #codes to execute
    #codes to execute
    #codes to execute
```

Keywords
One variable out of the container
Colon
A list of values (iterable)
4-Spaces / Tab (Indentations)

WHAT YOU JUST OBSERVED

- The **iteration variable** “iterates” through the **sequence** (ordered set)
- The **Action** of code is executed once for each value in the **sequence**
- The **iteration variable** moves through all of the values in the **sequence**



```
countdown = [5, 4, 3, 2, 1]

for i in countdown:
    print(i)
print('Peekaboo!')
```

A DEFINITE LOOP WITH STRING*

- The for loop construct works regardless of the data type in the list

```
friends = ['Joseph', 'Glenn', 'Sally']

for friend in friends :
   print('Happy New Year:' +  friend)

print('Done!')
```

Output:

```
Happy New Year: Joseph
Happy New Year: Glenn
Happy New Year: Sally
Done!
```

TYPES OF FUNDAMENTAL PROBLEMS WE CAN SOLVE

- The earlier questions involved just mindlessly accessing data and printing it, but the clever combination of iteration + condition, can allow us to introduce some intelligence to our repetition

- The following are the fundamental types of questions we can solve using a for loop (works for while loop as well, but its more compatible with for loop) and conditions , and we will explore them through a series of in-class practice:

    - Solving counting questions

    - Finding and returning specific elements that fulfils certain conditions

    - Aggregation or tally of statistics

PROBLEM SOLVING FRAMEWORK FOR DUMMIES (CITU)

- Given the sequence: [3, -4, 12, 9, -72, 0, 15], count the number of elements that are larger than 10 in the sequence

$$\boxed{1}\ \boxed{\textbf{Create an empty container to contain the results}} \longrightarrow \text{results = 0}$$

PROBLEM SOLVING FRAMEWORK FOR DUMMIES (CITU)

- Given the sequence: [3, -4, 12, 9, -72, 0, 15], count the <u>number of elements that are larger than 10</u> in the sequence

1 Create an empty container to contain the results → results = 0

2 Iterate thru the list using a for/while loop → for data in sequence:

PROBLEM SOLVING FRAMEWORK FOR DUMMIES (CITU)

- Given the sequence: [3, -4, 12, 9, -72, 0, 15], count the <u>number of elements that are larger than 10</u> in the sequence

1	Create an empty container to contain the results	→	`results = 0`
2	Iterate thru the list using a for/while loop	→	`for data in sequence:`
3	Test using conditions to see if value adheres to needs	→	`    if data > 10:`

```
results = 0
for data in sequence:
    if data > 10:
```

PROBLEM SOLVING FRAMEWORK FOR DUMMIES (CITU)

- Given the sequence: [3, -4, 12, 9, -72, 0, 15], count the <u>number of elements that are larger than 10</u> in the sequence

1	**C**reate an empty container to contain the results	`results = 0`
2	**I**terate thru the list using a for/while loop	`for data in sequence:`
3	**T**est using conditions to see if value adheres to needs	`if data > 10:`
4	**U**pdate the value in the results container	`results += 1`

```
results = 0
for data in sequence:
    if data > 10:
        results += 1
```

PROBLEM SOLVING FRAMEWORK FOR DUMMIES (CITU)

- Given the sequence: [3, -4, 12, 9, -72, 0, 15], count the <u>number of elements that are larger than 10</u> in the sequence

1. Create an empty container to contain the results
2. Iterate thru the list using a for/while loop
3. Test using conditions to see if value adheres to needs
4. Update the value in the results container

The **CITU / CIU** framework applies to most problem sets!!

This depends on whether there is a need to test conditions

SOLVING COUNTING QUESTIONS

- The question format and principle for solving counting questions are as follows:
    - Given the sequence: [3, -4, 12, 9, -72, 0, 15], count the number of positive numbers in the sequence
    - Thought process for solving this problem:

```
1. Create a counter/tally variable and set count to zero
2. Iterate through the list itself
3. Using a condition, check if the number is positive, if it is positive, I will increment the count by 1
4. Once the iteration end, I will print the counter to illustrate the total number of positive numbers in the sequence
```

IN-CLASS PRACTICE: UNDERSTANDING FOR LOOPS*

- Given the sequence: [3, -4, 12, 9, -72, 0, 15],
    - Count the number of positive numbers in the sequence
    - Count the number of negative numbers in the sequence

SOLVING SEARCHING QUESTIONS

- The question format and principle for solving counting questions are as follows:
  - Given the sequence: [3, -4, 12, 9, -72, 0, 15], return the largest positive number
  - Thought process for solving this problem:

1. **C**reate a highest_number variable and assign the first value of the list to it
2. **I**terate through the list itself
3. **T**est if the number is larger than the number contained within the highest_number variable
4. **U**pdate the highest_number variable with the current number if it is larger than the number stored in highest_number

IN-CLASS PRACTICE: UNDERSTANDING FOR LOOPS*

- Given the sequence: [3, -4, 12, 9, -72, 0, 15],
    - Find the biggest number in the sequence
    - Find the smallest number in the sequence

AGGREGATION & STATISTICAL TALLY QUESTIONS

- The question format and principle for solving counting questions are as follows:
    - Given the sequence: [3, -4, 12, 9, -72, 0, 15], return the aggregate of the sequence
    - Thought process for solving this problem:

1. **C**reate a total variable and set it to zero
2. **I**terate through the list itself
3. **U**pdate the total_variable by adding current number to total variable (total_variable += num)

IN-CLASS PRACTICE: UNDERSTANDING FOR LOOPS*

- Given the sequence: [3, -4, 12, 9, -72, 0, 15],
  - Find the sum of all the numbers in the sequence
  - Find the mean (average) of the sequence
  - *Hint: Combine the aggregation concept + the counting concept*

EXECUTION TRACE: SUM EXAMPLE

```
total = 0
data = [5, 4, -3, 2, 1]

for number in data:
    total += number
    print('Total =', total)
print('Final Total =', total)
```

Output:

1

EXECUTION TRACE: SUM EXAMPLE

```
total = 0
data = [5, 4, -3, 2, 1]

for number in data:
    total += number
    print('Total =', total)
print('Final Total =', total)
```

total:

Output:

2

EXECUTION TRACE: SUM EXAMPLE

```
total = 0
data = [5, 4, -3, 2, 1]

for number in data:
    total += number
    print('Total =', total)
print('Final Total =', total)
```

number 5

total:

Output:

3

EXECUTION TRACE: SUM EXAMPLE

```
total = 0
data = [5, 4, -3, 2, 1]

for number in data:
    total += number
    print('Total =', total)
print('Final Total =', total)
```

number 5

total:

+ (5)

Output:

4

EXECUTION TRACE: SUM EXAMPLE

```
total = 0
data = [5, 4, -3, 2, 1]

for number in data:
    total += number
    print('Total =', total)
print('Final Total =', total)
```

number 5

total:

```
+ (5)
```

Output:

```
Total = 5
```

5

EXECUTION TRACE: SUM EXAMPLE

```
total = 0
data = [5, 4, -3, 2, 1]

for number in data:
    total += number
    print('Total =', total)
print('Final Total =', total)
```

number `4`

total:

```
+ (5)
```

Output:

```
Total = 5
```

6

EXECUTION TRACE: SUM EXAMPLE

```
total = 0
data = [5, 4, -3, 2, 1]

for number in data:
    total += number
    print('Total =', total)
print('Final Total =', total)
```

number 4

total:

```
+ (5)
+ (4)
```

Output:

```
Total = 5
```

7

EXECUTION TRACE: SUM EXAMPLE

```
total = 0
data = [5, 4, -3, 2, 1]

for number in data:
    total += number
    print('Total =', total)
print('Final Total =', total)
```

number 4

total:

```
+ (5)
+ (4)
```

Output:

```
Total = 5
Total = 9
```

8

EXECUTION TRACE: SUM EXAMPLE

```
total = 0
data = [5, 4, -3, 2, 1]

for number in data:
      total += number
      print('Total =', total)
print('Final Total =', total)
```

number -3

total:

```
+ (5)
+ (4)
```

Output:

```
Total = 5
Total = 9
```

9

EXECUTION TRACE: SUM EXAMPLE

```
total = 0
data = [5, 4, -3, 2, 1]

for number in data:
    total += number
    print('Total =', total)
print('Final Total =', total)
```

number `-3`

total:

```
 +  (5)
 +  (4)
+  (-3)
```

Output:

```
Total = 5
Total = 9
```

10

EXECUTION TRACE: SUM EXAMPLE

```
total = 0
data = [5, 4, -3, 2, 1]

for number in data:
    total += number
    print('Total =', total)
print('Final Total =', total)
```

number -3

total:

```
+  (5)
+  (4)
+ (-3)
```

Output:

```
Total = 5
Total = 9
Total = 6
```

11

EXECUTION TRACE: SUM EXAMPLE

```
total = 0
data = [5, 4, -3, 2, 1]

for number in data:
    total += number
    print('Total =', total)
print('Final Total =', total)
```

number 2

total:

```
+  (5)
+  (4)
+ (-3)
```

Output:

```
Total = 5
Total = 9
Total = 6
```

12

EXECUTION TRACE: SUM EXAMPLE

```
total = 0
data = [5, 4, -3, 2, 1]

for number in data:
    total += number
    print('Total =', total)
print('Final Total =', total)
```

number 2

total:

```
 +  (5)
 +  (4)
 + (-3)
 +  (2)
```

Output:

```
Total = 5
Total = 9
Total = 6
```

13

EXECUTION TRACE: SUM EXAMPLE

```
total = 0
data = [5, 4, -3, 2, 1]

for number in data:
    total += number
    print('Total =', total)
print('Final Total =', total)
```

number 2

total:

```
+  (5)
+  (4)
+ (-3)
+  (2)
```

Output:

```
Total = 5
Total = 9
Total = 6
Total = 8
```

14

EXECUTION TRACE: SUM EXAMPLE

```
total = 0
data = [5, 4, -3, 2, 1]

for number in data:
    total += number
    print('Total =', total)
print('Final Total =', total)
```

number `1`

total:

```
+ (5)
+ (4)
+ (-3)
+ (2)
```

Output:

```
Total = 5
Total = 9
Total = 6
Total = 8
```

15

EXECUTION TRACE: SUM EXAMPLE

```python
total = 0
data = [5, 4, -3, 2, 1]

for number in data:
    total += number
    print('Total =', total)
print('Final Total =', total)
```

number `1`

total:

```
 +  (5)
 +  (4)
+  (-3)
 +  (2)
 +  (1)
```

Output:

```
Total = 5
Total = 9
Total = 6
Total = 8
```

16

EXECUTION TRACE: SUM EXAMPLE

```
total = 0
data = [5, 4, -3, 2, 1]

for number in data:
    total += number
    print('Total =', total)
print('Final Total =', total)
```

number 1

total:

```
 +  (5)
 +  (4)
+  (-3)
 +  (2)
 +  (1)
```

Output:

```
Total = 5
Total = 9
Total = 6
Total = 8
Total = 9
```

17

EXECUTION TRACE: SUM EXAMPLE

```
total = 0
data = [5, 4, -3, 2, 1]

for number in data:
    total += number
    print('Total =', total)
print('Final Total =', total)
```

number `1`

total:

```
 +  (5)
 +  (4)
+  (-3)
 +  (2)
 +  (1)
```

Output:

```
   Total = 5
   Total = 9
   Total = 6
   Total = 8
   Total = 9
Final Total = 9
```

18

OTHER WAYS TO IMPLEMENT A FOR LOOP

- Using the `range()` function in for loop

```
for i in range(0,5):
    print(i)
print("Peekaboo")
```

```
Output:
0
1
2
3
4
Peekaboo!
```

OTHER WAYS TO IMPLEMENT A FOR LOOP

```
for i in range(0,5):
   print(i)
print("Peekaboo")
```

Same as the following

```
the_list = [0,1,2,3,4]
for i in the_list :
   print(i)
print("Peekaboo")
```

- You can think of **`range(0,5)`** as a method that returns you a list: [0,1,2,3,4]
- Observe also that range does not include the last number

IN-CLASS PRACTICE: FILTERING LOOPS USING RANGE*

- Use the “for range” method to count and print the number of positive odd numbers in the following list: [3, -4, 12, 9, -72, 0, 15]

Hint:

1. *len() methods return the size of the list*
2. *Recall that using the index (i.e “list[0]”) function will return you a specific element for the list*

APPLYING THE FUNDAMENTALS TO DICTIONARY

- We learnt how to solve various fundamental types of questions we can solve using a for loop + conditions with list as the datatype. The same exact principle applies to dictionary, just that you have to do some extra

List

```
seq = [10,20,30,40]
large_num = seq[0]

for i in seq:
    if i > large_num:
        large_num = I

print(large_num)
```

Dictionary

```
seq = {'num_1':10,'num_2':20,'num_3':30}
large_num = seq['num_1']

for key,value in seq.items():
    if value > large_num:
        large_num = value

print(large_num)
```

APPLYING THE FUNDAMENTALS TO DICTIONARY

- Let’s understand it a bit more in detail

```
seq = {'num_1':10,'num_2':20,'num_3':30}

for key,value in seq.items():
    print('key =', key)
    print('value =',value)
    print('-----')
```

By using .items() on a dictionary, it returns two items to you each iteration, instead of the usual 1 items for a list

Key & value are just arbitrary variables I created to contain the two items returned

When I print the 2 of them, we will realise that it is actually just a key, and its associated value

APPLYING THE FUNDAMENTALS TO DICTIONARY

- Let’s understand it a bit more in detail

```
seq = {'num_1':10,'num_2':20,'num_3':30}

for key,value in seq.items():
    print('key =', key)
    print('value =',value)
    print('-----')
```

Output:

```
key = num_1
value = 10
-----
```

APPLYING THE FUNDAMENTALS TO DICTIONARY

- Let's understand it a bit more in detail

```
seq = {'num_1':10,'num_2':20,'num_3':30}

for key,value in seq.items():
    print('key =', key)
    print('value =',value)
    print('-----')
```

Output:

```
key = num_1
value = 10
-----
key = num_2
value = 20
-----
```

APPLYING THE FUNDAMENTALS TO DICTIONARY

- Let’s understand it a bit more in detail

```
seq = {'num_1':10,'num_2':20,'num_3':30}

for key,value in seq.items():
    print('key =', key)
    print('value =',value)
    print('-----')
```

Output:

```
key = num_1
value = 10
-----
key = num_2
value = 20
-----
key = num_3
value = 30
-----
```

FOOD FOR THOUGHT*

- So how do we for example find the sum of seq?

```python
seq = {'num_1':10,'num_2':20,'num_3':30}
```

SUMMARY

- Definite and Indefinite Loops
- for loops (definite)
  - Finding the largest/smallest
  - Counting
  - Summing
  - Average
  - Filtering results
- Range for loop
- Dictionary

WHILE LOOP (SELF STUDY)

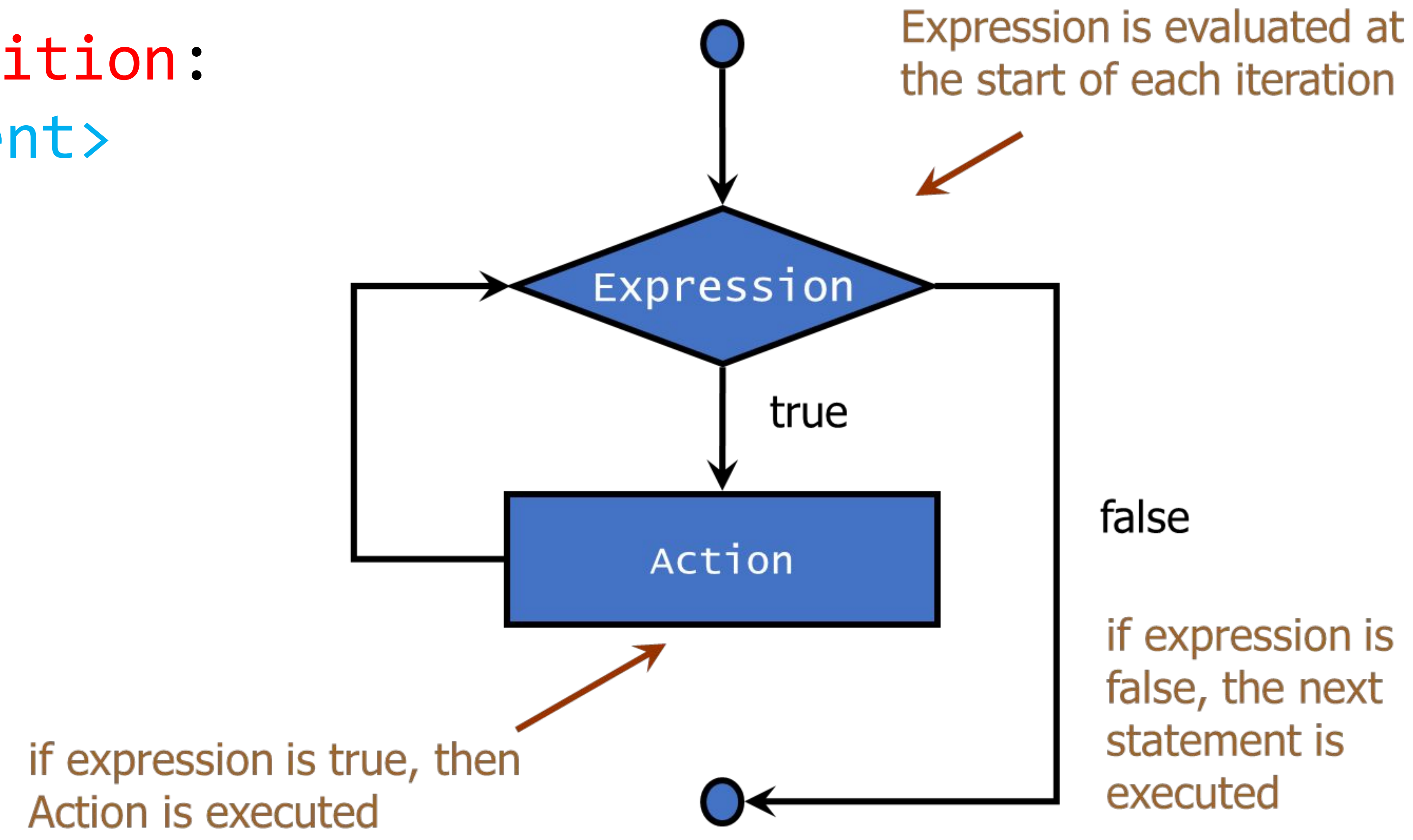
- Execution
- What are variable tasks?
- Types of iteration constructs (While & For)

INDEFINITE LOOPS

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

WHILE LOOP CONSTRUCT

```
while Condition:
    <Statement>
```


Expression is evaluated at the start of each iteration
Expression
true
Action
false
if expression is false, the next statement is executed
if expression is true, then Action is executed

WHILE LOOP CONSTRUCT

returns True OR False

```
while Condition:
    Statement
    Statement
    Statement
```

Statements will be executed if condition is true

WHILE LOOP: SIMPLE EXAMPLE*

```
while Condition:
    Statement
    Statement
    Statement
```

Try these two simple examples to convince yourself that all the while loop really needs is a True/False to work!

And as long as the condition is True, the action will always be performed, as along as the condition is False, nothing will ever happen

```
while True:
    print(1)
```

```
while False:
    print(1)
```

WHILE LOOP: TRUE / FALSE CONDITION TO WORK*

- Any conditions that return a true/false to the while construct would work

```
while Condition:
    Statement
    Statement
    Statement
```

```
i = 10
while i > 10:
    print(1)
```

```
i = 0
while i < 10:
    print(1)
```

You would realise that this loop is never ending though. How do we address this?

WHILE LOOP: CONDITIONS THAT ARE DEFINITE*

- Basically to end a loop, we create a condition that is only true for a limited amount of time

```
while Condition:
    Statement
    Statement
    Statement
```

```
i = 0
while i < 10:
    print(1)
    i += 1
```

As i increments every round, it means that the condition of i<10 does not hold true forever. This means that there will be an end to things!

EXECUTION TRACE: WHILE LOOP*

```
video_titles = [
    'despacito',
    'see_you_again',
    'im_yours'
]
num_of_videos = len(video_titles)
current_video_index = 0

while current_video_index < num_of_videos:
    print(video_titles[current_video_index])
    current_video_index += 1

print('that's all folks!')
```

output

EXECUTION TRACE: WHILE LOOP*

```
video_titles = [
    'despacito',
    'see_you_again',
    'im_yours'
]
num_of_videos = len(video_titles)
current_video_index = 0

while current_video_index < num_of_videos:
    print(video_titles[current_video_index])
    current_video_index += 1

print('that's all folks!')
```

num_of_videos 3

output

EXECUTION TRACE: WHILE LOOP*

```python
video_titles = [
    'despacito',
    'see_you_again',
    'im_yours'
]
num_of_videos = len(video_titles)
current_video_index = 0

while current_video_index < num_of_videos:
    print(video_titles[current_video_index])
    current_video_index += 1

print('that's all folks!')
```

num_of_videos	3
current_video_index	0

output

```
```

EXECUTION TRACE: WHILE LOOP*

```
video_titles = [
    'despacito',
    'see_you_again',
    'im_yours'
]
num_of_videos = len(video_titles)
current_video_index = 0

while current_video_index < num_of_videos:
    print(video_titles[current_video_index])
    current_video_index += 1

print('that's all folks!')
```

num_of_videos	3
current_video_index	0

output

```
```

True

EXECUTION TRACE: WHILE LOOP*

```
video_titles = [
    'despacito',
    'see_you_again',
    'im_yours'
]
num_of_videos = len(video_titles)
current_video_index = 0

while current_video_index < num_of_videos:
    print(video_titles[current_video_index])
    current_video_index += 1

print('that's all folks!')
```

num_of_videos	3
current_video_index	0

output

```
despacito
```

EXECUTION TRACE: WHILE LOOP*

```
video_titles = [
    'despacito',
    'see_you_again',
    'im_yours'
]
num_of_videos = len(video_titles)
current_video_index = 0

while current_video_index < num_of_videos:
    print(video_titles[current_video_index])
    current_video_index += 1

print('that's all folks!')
```

num_of_videos 3

current_video_index 1

output

```
despacito
```

EXECUTION TRACE: WHILE LOOP*

```
video_titles = [
    'despacito',
    'see_you_again',
    'im_yours'
]
num_of_videos = len(video_titles)
current_video_index = 0

while current_video_index < num_of_videos:
    print(video_titles[current_video_index])
    current_video_index += 1

print('that's all folks!')
```

num_of_videos	3
current_video_index	1

output

```
despacito
```

True

EXECUTION TRACE: WHILE LOOP*

```
video_titles = [
    'despacito',
    'see_you_again',
    'im_yours'
]
num_of_videos = len(video_titles)
current_video_index = 0

while current_video_index < num_of_videos:
    print(video_titles[current_video_index])
    current_video_index += 1

print('that's all folks!')
```

num_of_videos	3
current_video_index	1

output

```
see_you_again
```

EXECUTION TRACE: WHILE LOOP*

```
video_titles = [
    'despacito',
    'see_you_again',
    'im_yours'
]
num_of_videos = len(video_titles)
current_video_index = 0

while current_video_index < num_of_videos:
    print(video_titles[current_video_index])
    current_video_index += 1

print('that's all folks!')
```

num_of_videos	3
current_video_index	2

output

```
see_you_again
```

EXECUTION TRACE: WHILE LOOP*

```
video_titles = [
    'despacito',
    'see_you_again',
    'im_yours'
]
num_of_videos = len(video_titles)
current_video_index = 0

while current_video_index < num_of_videos:
    print(video_titles[current_video_index])
    current_video_index += 1

print('that's all folks!')
```

True

num_of_videos	3
current_video_index	2

output

```
see_you_again
```

EXECUTION TRACE: WHILE LOOP*

```
video_titles = [
    'despacito',
    'see_you_again',
    'im_yours'
]
num_of_videos = len(video_titles)
current_video_index = 0

while current_video_index < num_of_videos:
    print(video_titles[current_video_index])
    current_video_index += 1

print('that's all folks!')
```

num_of_videos	3
current_video_index	2

output

```
im_yours
```

EXECUTION TRACE: WHILE LOOP*

```python
video_titles = [
    'despacito',
    'see_you_again',
    'im_yours'
]
num_of_videos = len(video_titles)
current_video_index = 0

while current_video_index < num_of_videos:
   print(video_titles[current_video_index])
   current_video_index += 1

print('that's all folks!')
```

num_of_videos 3

current_video_index 3

output

```
im_yours
```

EXECUTION TRACE: WHILE LOOP*

```
video_titles = [
    'despacito',
    'see_you_again',
    'im_yours'
]
num_of_videos = len(video_titles)
current_video_index = 0

while current_video_index < num_of_videos:
    print(video_titles[current_video_index])
    current_video_index += 1

print('that's all folks!')
```

num_of_videos	3
current_video_index	3

output

```
im_yours
```

False

EXECUTION TRACE: WHILE LOOP*

```
video_titles = [
    'despacito',
    'see_you_again',
    'im_yours'
]
num_of_videos = len(video_titles)
current_video_index = 0

while current_video_index < num_of_videos:
    print(video_titles[current_video_index])
    current_video_index += 1

print('that's all folks!')
```

num_of_videos 3

current_video_index 3

output

```
that's all folks!
```