

Iteration I

A loop allows you to execute the same statements multiple times. Python has two kinds of loop structures: `for` loops, which iterate over the items of a sequence, and `while` loops, which continue to execute as long as a condition is true. This week we will study `for` loops.

Warm-up

1. Complete the following code that reads in scores separated by a comma and prints the minimum, maximum and average score.

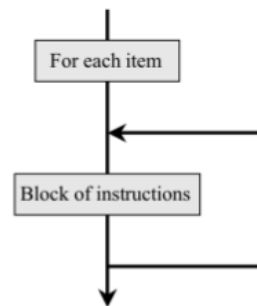
```
1 def get_stats(user_input: str) -> list:
2     str_list = user_input.split(',')
3     max_score = 0
4     min_score = 100
5     sum_scores = 0
6     for s in str_list:
7         score = float(s)
8         sum_scores += score
9         max_score = max(max_score, score)
10        min_score = min(min_score, score)
11
12    return [min_score, max_score, sum_scores/len(str_list)]
13
14 def main() -> None:
15     scores = input("Scores: ") # user inputs: "90.5,99.0,94.5,97.0"
16     stats = get_stats(scores)
17     print(stats) # expected output: [90.5,99.0,95.25]
18
19 main()
```

Model 1 `for` Statements

A `for` loop executes the same block of code “for each item in a sequence”. The instructor will trace `loops.py`, that contains the following code:

```
def main() -> None:
    print("hello")
    for x in [2, 7, 1]:
        print("the number is", x)
        print("goodbye")

main()
```



2. How many times does the indented line of code execute under the `for` loop? 3 times

3. How many times does the line of code NOT indented execute after the `for` loop? 1 time

4. Identify the value of `x` each time the indented line of code is executed.

a) 1st time: `x = 2`

b) 2nd time: `x = 7`

c) 3rd time: `x = 1`

5. Indicate how many times the `for` loop executes.

a) non-consecutive numbers: `[5, -7, 0]` 3 times

b) numbers decreasing in value: `[3, 2, 1, 0]` 4 times

c) all have the same value: `[4, 4]` 2 times

d) single value in a list: `[8]` 1 time

6. In general, what determines the number of times that the loop repeats?

The length of the list.

7. What determines the value of the variable `x`? Explain your answer in terms of what is assigned (`x = ...`) each time the loop runs.

The value `x` is selected from the list. Each time the loop runs, the next value from the list is assigned to `x`.

8. **Loop tables** are very useful when tracing the changes to the state inside of the loop. Draw the loop table corresponding to the code used in the demo:

<code>x</code>	Output
2	the number is 2
7	the number is 7
1	the number is 1
outside the loop	goodbye

9. Consider the following modifications to the program:

a) Write a statement that assigns `[0, 1, 2, 3, 4]` to the variable `numbers`.

```
numbers = [0, 1, 2, 3, 4]
```

b) Rewrite the `for x ...` statement to use the variable `numbers` instead.

```
for x in numbers:
```

c) Does the assignment need to come before or after the `for` statement?

Before

10. Consider the following code snippet:

```
for c in "Hi!":  
    print(c)
```

a) What is the output of this `for` statement?

```
H  
i  
!
```

b) What determined how many times `print(c)` was called?

The length of the string

c) Explain what a `for` statement does with strings.

It iterates over each character

11. What data types can and can't a `for` loop handle?

Sequences like lists and strings; Numbers don't work; you can loop over integers and floats.

Model 2 The `range` Function

The Python `range` function will generate a list of numbers. The `range` function can take up to three numbers as arguments. Fill in the table below:

Python code	Output
<code>range(5)</code>	<code>range(0, 5)</code>
<code>list(range(5))</code>	<code>[0, 1, 2, 3, 4]</code>
<code>x = range(3)</code>	
<code>print(x)</code>	<code>range(0, 3)</code>
<code>print(list(x))</code>	<code>[0, 1, 2]</code>
<code>list(range(5, 10))</code>	<code>[5, 6, 7, 8, 9]</code>
<code>list(range(-3, 4))</code>	<code>[-3, -2, -1, 0, 1, 2, 3]</code>
<code>list(range(4, 10, 2))</code>	<code>[4, 6, 8]</code>
<code>for i in range(5): print(i)</code>	prints 0, 1, 2, 3, 4 (separate lines)

12. Explain the difference in output between the first two lines of code (with and without the `list` function).

The first line of output describes the range as a function. The second line shows the actual range of values as a list.

13. If the argument of the `range` function specifies a single number (x):

- a) What will be the first number listed? `0`
- b) What will be the last number listed? `x - 1`
- c) How many numbers will be in the list? `x`
- d) Use the range function to generate the sequence 0, 1, 2, 3. `range(4)`

14. If the argument of the `range` function specifies two numbers (x, y):

- a) What will be the first number listed? `x`
- b) What will be the last number listed? `y - 1`
- c) How many numbers will be in the list? `y - x`
- d) Use the range function to generate the sequence 1, 2, 3, 4. `range(1, 5)`

15. If the argument of the `range` function specifies three numbers (x, y, z):

- a) What will be the first number listed? `x`
- b) What does the third argument represent? `how much to add each time`
- c) How many numbers will be in the list? `[(y - x) / z]`
- d) Use the range function to generate the sequence 1, 3, 5, 7. `range(1, 8, 2)` or `range(1, 9, 2)`

16. Modify the `for` statement in Model 1 so that the number of times the loop executes is determined by a variable named `times`.

a) How did you change the `for` statement?

```
for i in range(times): # no need for list() conversion
```

b) How would you cause the loop to print the values 0 to 5?

```
Add this line before the loop: times = 6
```

17. Consider the two different types of `for` statements used in Model 1 and 2.

a) If you wanted to execute a loop 100 times, which type of `for` statement would you choose and why?

```
for i in range(number), so that you don't have to specify the list.
```

- b) If you wanted to use each item of an existing list inside the loop, which type of `for` statement would you choose and why?

`for i in list`, since the list exists already and might not be a range.

18. Does the `range` function work with strings? If so, show an example. If not, show how to print the letters A to Z in a loop.

The arguments to `range` must be integers. You can use the built-in function `chr` to convert integers to their corresponding Unicode characters:

```
for i in range(65, 91):  
    print(chr(i))
```

Model 3 The Accumulator Pattern

The pattern of iterating the updating of a variable is commonly referred to as the *accumulator pattern*. We refer to the variable as the *accumulator*. The anatomy of the accumulation pattern includes:

- *initializing an accumulator variable* to an initial value (such as 0 if accumulating a sum).
- *iterating* (e.g., traversing the items in a sequence).
- *updating the accumulator variable* on each iteration (i.e., when processing each item in the sequence).

Consider the following code snippet:

```
1 mystery = "@4b!"  
2 count = 0  
3 for w in mystery:  
4     count = count + 1  
5 print(count)
```

19. Which line of code initializes the accumulator variable? 2

20. Which line of code is iterating? 3

21. Which line of code updates the accumulator variable? 4

22. What is the purpose of this code snippet?

It counts how many characters are in the string `mystery`.

23. Draw the loop table corresponding to this code snippet:

w	count
'@'	1
'4'	2
'b'	3
'!'	4

24. Write a for loop that makes a copy of `mystery` by copying each character one-by-one and then prints the result:

```

1 mystery = "@4b!"
2 copy = ""
3 for c in mystery:
4     copy = copy + c
5 print(copy)

```

25. Draw the loop table corresponding to the previous code snippet:

c	copy
'@'	'@'
'4'	'@4'
'b'	'@4b'
'!'	'@4b!'

26. Write a for loop that reverses the characters in `mystery` and then prints the result:

```

1 mystery = "@4b!"
2 reversed = ""
3 for c in mystery:
4     reversed = c + reversed
5 print(reversed)

```

27. Draw the loop table corresponding to the previous code snippet:

c	reversed
'@'	'@'
'4'	'4@'
'b'	'b4@'
'!'	'!b4@'