Sequences I and Iteration I

Model 1 Sequence data types: strings and lists

So far we learned about variable data types that can only hold one value, namely, `int` and `float`. Strings and lists are examples of ***sequence*** data types. Just as a string variable can hold multiple characters, a variable of type list can hold multiple values in the form of a ***list***. The values are separated by commas and wrapped in square brackets. For example:

```
primes = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
```

1. Complete the table below to explore how sequences work:

Python code	Output
`seq1 = "one two"`	
`type(seq1)`	
`len(seq1)`	
`seq3 = ["one", "two"]`	
`type(seq3)`	
`len(seq3)`	

The word "sequence" is a generic term for an ordered set of data. Each ***element*** of the list can be referenced by an ***index***, which is the sequential position starting at 0. For example, `primes[4]` is 11.

index	0	1	2	3	4	5	6	7	8	9
value	2	3	5	7	11	13	17	19	23	29

2. Evaluate each line of code and write down its corresponding output. If an error occurs, write what type of error. Place an asterisk (*) next to any output for which you are not sure.

Python code	Output
`odd = [1, 3, 5, 7]`	
`print(odd)`	
`print(odd[2])`	
`len(odd)`	
`number = 12345`	
`print(number[3])`	

3. What is the index of the second element of `primes`? What is the value at that index?

4. How does the index number compare to the position of the element?

5. How did you reference the value of the 3rd element of `odd`?

6. What did the output of the len() function tell you about the list?

7. Write a statement that assigns a list of three integers to the variable `run`.

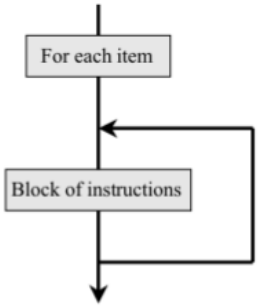
8. How does a sequence type differ from a number? (See the last row of the table.)

Model 2 `for` Statements

A loop allows you to execute the same statements multiple times (therefore, we also say that a loop is a repetition structure). 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.
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()
```



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

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

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

a) 1st time:

b) 2nd time:

c) 3rd time:

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

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

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

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

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

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

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

15. ***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:

x	Output

16. Consider the following modifications to the program:

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

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

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

17. Consider the following code snippet:

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

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

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

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

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

Model 3 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
`range(5)`	
`list(range(5))`	
`x = range(3)`	
`print(x)`	
`print(list(x))`	
`list(range(5, 10))`	
`list(range(-3, 4))`	
`list(range(4, 10, 2))`	
`for i in range(5):` `  print(i)`	

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

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

a) What will be the first number listed?

b) What will be the last number listed?

c) How many numbers will be in the list?

d) Use the range function to generate the sequence 0, 1, 2, 3.

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

a) What will be the first number listed?

b) What will be the last number listed?

c) How many numbers will be in the list?

d) Use the range function to generate the sequence 1, 2, 3, 4.

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

a) What will be the first number listed?

b) What does the third argument represent?

c) How many numbers will be in the list?

d) Use the range function to generate the sequence 1, 3, 5, 7.

23. 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?

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

24. 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?

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?

25. 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.

Model 4 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)
```

26. Which line of code initializes the accumulator variable?

27. Which line of code is iterating?

28. Which line of code updates the accumulator variable?

29. What is the purpose of this code snippet?

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

w	count

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