# **Warm-Up #4: Loops and Lists**

| **Understanding Repetition Statements - while** |
| --- |

A **while** loop requires a boolean expression to be **True** in order for the code inside of the body of the loop to repeat.

| num = input("Enter a non-negative number") **while** num < 0:  num = input("Enter a non-negative number")  print(f"You entered {num}, which is not a negative number") |
| --- |

In this example, the loop will continue to prompt the user for a non-negative number until the user enters a number which causes num < 0 to evaluate to **False**. We have no way of knowing how many times a user will enter a negative number.

**while** loops are called ***variable repetition*** loops because they are useful when there is no easy way to predict how many times the loop will need to repeat from looking at the Boolean expression contained within the while loop header.

| **Understanding Repetition Statements - for** |
| --- |

**for** loops are known as **fixed-repetition** loops because they tend to repeat a predictable number of times.

Python **for** loops are used to iterate over each items in an iterable collection of data (like list or str):

| **for** l **in** 'alpaca':  print(l) | **for** n **in** [11, 13, 17, 23, 29, 31]:  print(n) |
| --- | --- |
| 'a' 'l' 'p' 'a' 'c' 'a' | *11 13 17 23 29 31* |

| **Ordered Sequences- list and str** |
| --- |

A list is similar to a dynamic array in other languages. The elements in both a list and str are kept in a specific order, so list and str are both Python **ordered sequences**. Consequently, list and str use **index** notation (ie. [ ])to refer to specific items in the sequence.

### 

**Note**: Python allows for the use of negative indices.

### **Sequence Slicing**

| *pet = 'alpaca' #str print(pet[0]) # a print(pet[1]) # l print(pet[-1]) # a*  *print(pet[2:5]) # pac print(pet[2:]) # paca print(pet[::2]) # apc print(pet[::-1]) # acapla* | *favorite\_primes = [11, 13, 17, 23, 29, 31] #list print(favorite\_primes[0]) # 11 print(favorite\_primes[1]) # 13 print(favorite\_primes[-1]) #31*  *print(favorite\_primes[2:5]) # [17, 23, 29] print(favorite\_primes[2:]) # [17, 23, 29, 31] print(favorite\_primes[::2]) # [11, 27, 29] print(favorite\_primes[::-1]) # [31, 29, 23, 17, 13, 11]* |
| --- | --- |

Sequence slicing uses 1 - 3 parameters to generate a subsequence of the items in an ordered collection

* **sequence[2]**: The item in the third spot in the sequence is returned
* **sequence[2:5]**: Items in indices 2, 3, 4 are returned. The terminal index is not included (ie. exclusive)
* **sequence[1:7:2]**: Items in indices 1, 3, 5 are returned. 2 is the step size of the slicing.
* **sequence[10:0:-2]**: Items in indices 10, 8, 6, 4, 2 are returned (in that order).

### **Finding the length of a list/string**

| len(pet) *# number of letters in the string*  len(favorite\_primes) *# number of items in the list* |
| --- |

### **Inserting a new element at the end of a list**

| nums.append(10) *# insert 10 to the end of the list* |
| --- |

**Note**: Python strings are immutable so it’s not possible to modify the letters in a string. This includes appending to extend a string.

### **Accessing the index of a list/string in a for loop**

for loops can incorporate the index in an ordered sequence via the enumerate function:

| **for** index, item **in** enumerate(pet):  print(index, item) | **for** index, item **in** enumerate(favorite\_primes):  print(index, item) |
| --- | --- |
| 0, 'a' 1, 'l' 2, 'p' 3, 'a' 4, 'c' 5, 'a' | 0, 11 1, 13 2, 17 3, 23 4, 29 5, 31 |

| **Ordered Sequence- range** |
| --- |

A **range** is an immutable ordered sequence of integers. These examples use a **for** loop to iterate over the elements in a **range**:

| **for i in range(5):  print(i)** | **for i in range(5, 10):  print(i)** | **for i in (10, 20, 2):  print(i)** | **for i in range(10, 0, -2):   print(i)** |
| --- | --- | --- | --- |
| **0**  **1**  **2**  **3**  **4** | **5**  **6**  **7**  **8**  **9** | **10**  **12**  **14**  **16**  **18** | **10**  **8**  **6**  **4**  **2** |

The **for** loop will encounter each integer in the sequence, one integer at a time. The identifier in a **for** statement provides access to each successive integer in the **range**. The **range** of integers dictates the number of times that the loop will repeat.

The **range** function uses 1 - 3 parameters to generate a sequence of integers:

* **range(5)**: 5 is the terminal integer of the range and is not included in the range
* **range(5, 10)**: 5 is the first integer in the range. 10 is the terminal value of the range and is not included in the range
* **range(10, 20, 2)**: 10 is the first integer in the range. 20 is the terminal value of the range and is not included in the range. 2 is the step size of the range
* **range(10, 0, -2)**: 10 is the first integer in the range. 0 is the terminal value of the range and is not included in the range. -2 is the step size of the range

| **Generating Ordered Collections of Data- List Comprehensions** |
| --- |

Python **list comprehensions** offer a more concise way of generating a list of values from a given range:

| nums = [2\*i **for** i **in** range(0, 5)]  print(nums) *# [0, 2, 4, 6, 8]* | | nums = [i\*i **for** i **in** range(0, 10, 2)]  print(nums) *# [0, 4, 16, 36, 64]* |
| --- | --- | --- |
| nums = [x **for** x **in** range(10) **if** x % 2 == 0]  print(nums) *# [0, 2, 4, 6, 8]* | | nums = [0.5\*i **for** i **in** range(0, 5, 1)]  print(nums) *# [0, 0.5, 1, 1.5, 2.0]* |

| **Warm-Up #4 Exercises -** |
| --- |

### **Exercise 4.1: Implement the functions in** [**warm\_up\_4.py**](https://drive.google.com/file/d/15GqnQpGGB13zyjOl9k0iDTv2PHGIFk3a/view?usp=drive_link) **to see for loops and lists in action.**

Use loops and lists to generate a variety of integer sequences via menu-driven input. Be sure to consider edge cases when you implement each of the functions…

Submit your completed warm\_up\_4.py to the Warm-Up #4 assignment on Gradescope.

If you find that you’ve invented a new sequence of integers, check the [On-line Encyclopedia of Integer Sequences](https://oeis.org/) to see if somebody beat you to it.